

*Dennis Bathory Kitz*

# ***THE CUSTOM TRS-80 & OTHER MYSTERIES***



The complete guide to customizing TRS-80 software and hardware

*Dennis Bathory Kitsz*  
**THE CUSTOM TRS-80**  
**& OTHER MYSTERIES**

**David Moore — Editor**

**Thomas Nelson — Artistic Direction and Backup Vocals**

**Steve Rick and Charles Trapp — Graphics and Figures**

**H. C. Pennington — Cover Design**

**First Edition  
First Printing  
February 1982  
Printed in the United States of America.  
Copyright © 1982 by Dennis Bathory Kitsz**

**ISBN 0 936200 02 2**

All rights reserved. No Part of this book may be reproduced by any means without the express written permission of the publisher. Example programs are for personal use only. Every reasonable effort has been made to ensure accuracy throughout this book, but neither the author or publisher can assume responsibility for any errors or omissions. No liability is assumed for any direct, or indirect, damages resulting from the use of information contained herein.



Published by  
**IJG Inc**  
1260 West Foothill Blvd.,  
Upland, CA 91786, USA  
714 - 946 5805

**Radio Shack and TRS-80 are registered trademarks of the Tandy Corporation. Supplements to Chapters in this book have appeared in part in The Alternate Source, 1806 Ada Street, Lansing, Michigan 48910.**

# IMPORTANT

## Read This Notice

Any warranty that may exist on any TRS-80 computer hardware product that has been modified, altered, or has the seal broken is VOID.

Any software or computer hardware modifications are done at your own risk. Neither the PUBLISHER nor the AUTHOR assumes any responsibility or liability for loss or damages caused or alleged to be caused directly or indirectly by applying any modification or alteration to software or hardware described in this book, including but not limited to any interruption of service, loss of business, anticipatory profits or consequential damages resulting from the use or operation of such modified or altered computer hardware or software. Also, no patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the PUBLISHER and the AUTHOR assume no responsibility for errors or omissions.

The reader is the sole judge of his or her own skill and ability to perform the modifications and/or alterations contained in this book.

The below statement is the apparent policy of the TANDY CORP. and its subsidiary RADIO SHACK as understood by the publisher. The following statement is not endorsed by the TANDY CORP., but is presented for the reader's information and as a warning regarding the consequences of modifying or altering the TRS-80 computer hardware in any way. IJG, Inc., assumes no responsibility for the accuracy of the following statement.

The reader is warned that it is the apparent policy of the TANDY CORP. and its subsidiary RADIO SHACK to refuse repair service on any TRS-80 computer product that has been modified or altered in any way whatsoever.

However, a Radio Shack Computer Service Center, at the discretion of the individual Service Center, may repair an altered or modified TRS-80 computer upon the customer's prior agreement to pay any additional charges that may be incurred to remove or repair any and all alterations or modifications that may exist and thereby return the computer hardware to factory specifications.

---

## Editor's Note

---

### ABOUT THE AUTHOR

Dennis Bathory Kitz is a composer and active defender of the contemporary arts. However, this approach to life has resulted in pecuniary disaster which could only be ameliorated by working occasionally as a librarian, book editor, printing press operator, truck driver, newspaper editor, laborer, secretary, graphics designer, electronics technician, and typist. He is currently the director of the Dashuki Music Theatre, the president of Green Mountain Micro, and a regular columnist for several microcomputing magazines. He lives in Roxbury, Vermont with his wife Claire Manfredonia, as well as Aida, Mehitabel and Smokimoto (the cats), Fritz (the dog), and Fred and Ethel (the finches).

I'd like to say a word of thanks to Dennis Kitz for helping us through the entire range of Murphy's laws that were proven, over and over again, in the preparation of this book. . .

I'd also like to thank:  
Jim Perry, for teaching me to swim the old-fashioned way;  
Jim Murphy of Bishop Graphics, for his typing ability;  
Thomas Scott Nelson, for wandering through at a good time;  
Nancy DeDiemar, for the kind of help you don't find anymore;  
H.C. Pennington, for lending a hand with the impossible parts;  
Bruce Stuart, for lending a hand with Harv;  
and Charles Trapp for sticking with me to the end . . . which kept refusing to arrive.

And for those of you who have waited so long for this book, we believe you'll find the wait was worthwhile. There have been many updates of information that couldn't have been included a year ago, and several extras — including an additional chapter providing 111 cures for the common crash.

David E. Moore  
February 1982

## Contents

<b>Preface</b> .....	6	<b>Chapter 3:</b>	
<b>Acknowledgments</b> .....	7	Software Modifications .....	57
<b>Introduction:</b>		Sophisticated Debouncing .....	58
The Tools You Will Need .....	8	Upper/Lower Case Driver .....	59
Building a Power Supply .....	13	Box / Using the Editor/Assembler .....	60
<b>Chapter 1:</b>		Patching the BASIC Interpreter .....	62
Getting Inside .....	19	Box / Creating BASIC Tokens .....	63
What You See .....	19	Packing BASIC with Machine Code ....	66
Hesitation .....	20	Sound and Sound-Effects Generation ...	70
So Open It Already! .....	21	What's in a List? .....	70
Put It Back .....	23	Auto-Execution of BASIC Programs ....	71
The Hidden Insides .....	23	Machine Language Monitor .....	72
		Undoing NEW .....	73
Supplement to Chapter 1:		Resetting MEMORY SIZE? .....	74
Power-Up Routines of the TRS-80 .....	25	Peek That Keyboard .....	75
		The Make-'Em-Sweat Memory Test ....	77
<b>Chapter 2:</b>			
Copacetic Comprehension .....	31	Supplement to Chapter 3:	
Number Systems .....	31	On Cassette Input/Output .....	78
Box / Converting Binary to Decimal ....	32	Voice I/O .....	83
Box / Reading the Pins .....	33	<b>Chapter 4:</b>	
Digital Logic Devices .....	34	Simple Hardware Modifications .....	85
Into Machine Language .....	36	Expanding the Memory .....	86
What's in the Memory Map? .....	38	Box / Keep It Clean! .....	86
Setting MEMORY SIZE? .....	40	Box / Opening and Closing the Case ....	87
Level I vs Level II vs Level III .....	40	Box / Handling Integrated Circuits .....	89
Important Hardware Areas .....	42	Rescuing the Reset .....	91
Peripheral Addressing .....	51	Up-Front Reset .....	92
Box / Wire-Wrapping Technique .....	51	Working by the Woodstove .....	93
Box / Soldering Technique .....	52	Box / Making It Look Manufactured ....	96
		Working by the Woodstove - II .....	96
Supplement to Chapter 2:		Hexadecimal Keypad .....	98
On the Keyboard Scan .....	53	Reversing the Video .....	103
		Box / Clumsy? Me Too Do This First ....	105

## Contents

Lower Case with Upper .....	106	<b>Chapter 10:</b>	
One by One (Individual Reverse Video) ...	106	And Now It's Broken .....	229
Box / Carpentry Considerations II .....	107	Box / Replacing the Keyboard Cable ...	229
Stepping on the Accelerator .....	110	Home and Business Environments .....	231
Level I and Level II Together .....	112	When the Memory Crashes .....	232
Supplement to Chapter 4:		Box / Cleaning the Edge Connectors ...	233
On Relocatable Code .....	115	Box / Using an Oscilloscope .....	234
<b>Chapter 5:</b>		Aligning the Video .....	235
How the System Expands .....	121	Routine Maintenance .....	235
Radio Shack's Expansion Interface ....	121	Care of Peripherals .....	235
The Expansion Box Ground Problem ....	123	Machine Language Diagnosis Loops ....	236
Expanding to 32K and 48K Memory .....	124	<b>Chapter 11: 111 Cures for the Common</b>	
Speeding Up Newer Expansion Boxes ....	126	<b>Crash</b>	
LNW and Microtek Expansion Boxes ....	126	Common Problems and Their Cures ....	241
Box / Using Those 4K Leftovers .....	127	I. SOFTWARE .....	241
Special Section: Two Other 80s .....	129	II. POWER .....	242
Supplement to Chapter 5:		III. EDGE CONNECTORS .....	243
About Interrupts .....	134	IV. MEMORY .....	244
<b>Chapter 6:</b>		V. FIRMWARE .....	245
More Hardware Modifications .....	141	VI. RS-232 .....	246
Making HALT Work .....	141	VII. DISK .....	246
Fixing the Stuck Relay .....	142	VIII. TAPE .....	249
High-Resolution Graphics .....	143	IX. HARDWARE .....	250
BASIC ROMs Replaced by EPROMs ....	152	X. KEYBOARD .....	253
Powering Up to a Monitor on the Side ....	153	XI. VIDEO .....	253
Box / How Interpreters Work .....	154	XII. RFI .....	255
Supplement to Chapter 6:		XIII. HEAT .....	255
The CLOAD Problem .....	156	XIV. PRINTER .....	255
<b>Chapter 7:</b>		XV. USER PROCEDURE .....	255
Controlling the World .....	161	Last Thoughts .....	257
Controlling Electrical Appliances .....	162	<b>Index</b> .....	259
Analog-to-Digital, Digital-to-Analog ....	164	<b>Appendix I:</b>	
Making Music and Sound Effects .....	166	Parts Suppliers .....	267
<b>Chapter 8:</b>		<b>Appendix II:</b>	
Adding to the System .....	175	Bibliography .....	268
Parallel Printer Interface .....	175	<b>Appendix III:</b>	
Talking With the World —		ASCII Codes and Conversion Tables ...	271
The Computer as Boss .....	177	<b>Appendix IV:</b>	
Real-Time Clock/Calendars .....	182	KEEPIT 3.2 .....	277
Box / Edge Card Connectors: What's Up? ...	187	<b>Appendix V:</b>	
Bank Selecting Machine Code in ROM ...	189	Chip Pinouts .....	283
Box / The Romplus Board .....	198	<b>Appendix VI:</b>	
The Micro Front Panel Monitor .....	199	Glossary .....	313
<b>Chapter 9:</b>		<b>Appendix VII:</b>	
Keeping It Safe: Mass Storage .....	203	Parts lists & PC Boards .....	323
Disk Drives .....	204		
A Pictorial Tour of A Disk Drive .....	205		
A Heavy Dose of DOSes .....	209		
Box / A Garden Full of Varieties .....	209		
Exatron Stringy-Floppy .....	210		
Pictorial Tour of the Stringy-Floppy ...	210		
Fastload and Other Systems .....	211		
High Speed Cassette Loading .....	213		
A Paper Tape Reader .....	214		
An 8-Track Mass Storage System .....	217		

## Preface

---

A few years ago, when CB was king, a company riding high with those 10-4's hesitantly announced a very expensive new product. A few typed sheets with hand drawings were all store sales personnel had to explain this small 'home and business computer'. Neither Tandy nor we few first customers had any idea that the TRS-80 would become the first true home computer, the Model T of the microprocessor age.

There's no question that the 'Trash-80' carries the burden of some weak engineering and corner-cutting decisions made at the time of the CB decline, when this personal computer could have been Radio Shack's Waterloo instead of its new Wunderkind. But, like Rabelais's Gargantua, the new child soon outgrew its expectations – as well as its clothes.

And so everyone from the inner sanctum of the Tandy Corporation to the cluttered backrooms of a thousand hobbyists began the attempt to keep this new child content and amused. Thus was born the custom TRS-80. From the very beginning, and from seemingly nowhere, came forth educated hordes who would prod, poke, paint, primp, and prime this humble computer into becoming more than itself – a five-hundred-dollar supercomputer.

The machine couldn't always do it. Tandy took the rap, but also took the cash. Bruised and broken 80's littered the electronic landscape, yet also unrecognizably modified TRS-things took on tasks as diverse as business accounting, industrial control, and music making.

In this book, pathways to creating your own customized TRS-80 will be explored. The machine will be added to, opened and altered, its software patched, its uses expanded, and its breakdowns cured. If you expect that your personal computer can do more than a fancy game of video violence, then I hope you will join me in these explorations.

As this book was completed, Tandy announced that the TRS-80 – now called the Model I – is history. Production has ended, and that's a good

thing. The pressure is now off; there is no more Trash-80 to defend, but there still is a TRS-80 to put to use.

This book has been put together to please everyone. Of course it will not. Although I can't expect to help lead each of you through the intricacies of a TRS-80, I hope that, whether your wish is to jump right in and solder every wire or to learn slowly and deliberately the theory and practice of the machine, you can gain some insight from this volume.

Chapters have been arranged with basic theory and concepts toward the front. Special sections discussing the computer's software have been included to add some dimension to the concepts, and some of my own opinions, thoughts and tirades have been boxed throughout the text.

I have seen and used nearly all the commercial products presented in this book; comments, therefore, are based on first-hand knowledge, unless otherwise noted.

I have attempted in the appendices to present lists of those suppliers, publishers, terms, etc., that every TRS-80 user might want to know and not know how to find.

As the TRS-80 joins the computer museum along with the ENIAC, UNIVAC, IBM 370, and others, it still holds a more special place than many of those – with possibly half of million of its kin in use in the United States, Canada, Great Britain, Australia, Germany, and other countries. With that in mind, this book may need an occasional update.

I would appreciate receiving updated information, corrections, suggestions and criticism. Though I cannot promise a personal answer, those suggestions will be reflected in future printings of this book.

**Dennis Bathory Kitsz**

Roxbury, Vermont

February 1981

---

## Acknowledgments

Someday the interdependence and cooperation among authors and programmers in this infant field will be chronicled. I can think of a hundred books or programs or articles or newsletters, each one of which gave me an essential part of the insight needed to create this volume, and without any single one of which that creation would have been impossible. Among the programmers, authors, and friends . . .

Philip K. Hooper the Codesmith, *for teaching me the simplicity and elegance of machine code*; Ron Gillen of Lab Service, Inc., *for keeping me awash in new information*; Nick "Spike" Maggio of the Philadelphia/Castor Avenue Radio Shack, *for risking life, limb and manag-ership on Tandy's red tape battlefield, and for teaching me ventriloquism*; Roger Fuller of Fuller Software and Bryan Mumford of Mumford Micro Systems, *for two reference books I couldn't do without*; Jim Perry, *for his entrepreneurial acumen and refreshing lack of humility*; Michael Comendul of 80 Microcomputing, *for pointing out that English is a difficult second language for most programmers*; Debra Marshall of the same publication, *for pointing out that it doesn't have to be that way*; Dave Moore and Thomas Scott Nelson of IJG, *for saving this book from disaster*; Charley Butler and Joni Kosloski of The Alternate Source, *for my author's carte blanche*; Dave Beetle of the pioneer-

ing On-Line, *for really starting it all*; Harv Pennington *for jollies*; and for many and varied favors: Bill Johnson (*Cleveland Users Group*), Vaughn Jupe, Ron Troxell (*Personal Micro Computers*), Don Stoner (*The Peripheral People*), Lee Perryman (*The Associated Press*), Wayne Green (*himself*), Eric Maloney (*Kilobaud*), Jack Decker, Mike Barton (*MSB Electronics*), Leo Waltz, Jerry Sabin, Fred Blechman, Don Stevens, Walt Auch III, Vince Schulz, Bill Archbold (*Archbold Electronics*), Al Abrahamson (*Norwalk Users Group*), Bill Barden, Don Valentine (*Tecmark Associates*), Harold J. Matts, Stan Ockers, Thomas Frederick (*ABS Suppliers*), Les Logan (*TCS Newsletter*), Brian Harron (*Ottawa Users Group*), John Bilotta, Gregg Shadel, Don C. Tatum (*Barre-Montpelier Radio Shack*), Andrew Law, *the many manufacturers represented herein, and the usual coterie of others forgotten and maligned. Special thanks to the anonymous author of the TRS-80 Technical Reference Manual for a job well done.*

This book is dedicated to my wife, Claire Manfredonia, because she'll do most anything for a good gag. One day she walked into a serious meeting at an engineering company and hit me in the face with a blueberry pie. I won't tell you about the marshmallow fluff.

d.b.k.

March 1982

## Introduction

---

### The Tools You Will Need

Your basic TRS-80, with some attachments and software, is a thousand-dollar item. So I'll not encourage you to use dime-store tools. Buy the best you can afford, keep them clean, and reserve them just for use on the '80. Don't double up tools with the family auto. You may not need them all, but here is my customizer's toolbox:

**A medium-sized flat-blade screwdriver and Phillips blade screwdriver** (a reversible combination is ideal). With these you open cases and remove cabinets.

**A jeweler's set** of flat and Phillips blade screwdrivers; **hex nut drivers** are optional. These drivers can be used to align tape heads, help make delicate wire bends, adjust trimmer controls and even repair watches.

**One very thin screwdriver** for lifting integrated circuits out of sockets. This will be its only purpose, but the first time you break the pins off a \$10 jumper cable, you'll wish you'd used it!

**Small scissor-type cutters** (manicuring types are excellent). These will be used for snipping leads in tight spots.

**Small diagonal wire cutters** and/or front-cutting 'nippers'. Your general purpose cutters. They are fast and easy to use, but not to be used for heavy wire around the house.

**Needlenose pliers** (two pairs, normal and 90-degree types). You'll need these for bending leads, also extracting bits and pieces you've dropped into a nest of wiring.

**An X-acto type knife**, with a strong blade and handle you feel comfortable with. Since this will be used to cut delicate solder traces, you should be able to handle it deftly. I use a single edged razor blade, but have leather fingers!

**A scalpel**, if you can get one. For very delicate trimming and scraping; **a dental pick** for pulling off solder balls or lifting parts off a board (get this item from an obliging dentist — they are often discarded when worn); **tweezers and needlepoint hooks**. The latter come in handy for tracing incorrect wire-wrapped connections.

**Rat-tail, triangular, and flat files**. These are only for sprucing up the cosmetics, so if you don't care how it looks, save a few bucks.

**A wire-wrapping tool**. The decision on this can be tough. If you can afford it, get one of the electrically operated slit-and-wrap types. Stay away from 'just wrap' tools, since they depend on the sharpness and quality of the sockets; also they are useless for wrapping capacitors or resistors. I use a simple double-ended tool sold by Radio Shack for about \$5. It wears out after a thousand or so connections, but it fits my hand well, and is not clumsy like some electric units.

**A soldering iron**. The decision is not easy. Should you spend top dollar and get an expensive one or buy a cheap unit that can be discarded when it wears out? I use a \$5

To help you prepare for each project, the following graphic symbols have been used as a key to the tools needed for each project:



**Phillips screwdriver**



**Flat-blade screwdriver**



**Sockets**



**Thin lifting tool**



**Scissors type cutters**



**Solder**



**Wire cutters**



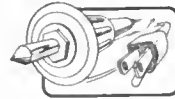
**Needlenose pliers**



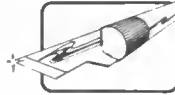
**Various files**



**Wire wrapping tool**



**Soldering iron**



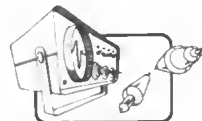
**X-acto knife or blade**



**Solder wick**



**Multimeter**



**Oscilloscope**



**Drill**

soldering iron which can be junked when it gets beat, but my editor uses the best he can get (a \$30 temperature-controlled one).

I file a set of \$1 tips to my satisfaction, and lubricate the threads with white heat sink grease. This way I have a few different tips at my disposal; with plated bits you **never** file the tips.

**A Multimeter.** The voltage regulators in your TRS-80 are very good, so any problems will usually show up as gross errors. This offers you a way out of buying an expensive multimeter; for most of these projects, the \$10 pocket variety will suffice.

However, for lots of repair work a better meter is in order; I use a \$40 type (not digital!) for my work.

**An oscilloscope.** For the projects, no. But for repairs, yes. Don't panic thinking of a thousand dollars for a digital scope, because an old color television scope will do perfectly well; they can be found in the bargain bins for \$50 to \$100. If it saves you

a \$100 repair bill, you've paid for it. Mine is an old RCA type WO-90Q, built for early color TV, and just fine for the bulk of TRS-80 work.

You will also need supplies in the tool box. Among these are:

**Solder.** Get the best you can afford. There's nothing so unpleasant as a great glob of the stuff between two traces on a board. Order the multicore rosin flux type, and stay away from most the off-the-shelf stuff. Remember, multicore rosin type only, and the finer the gauge the better. **Never use acid flux solder**, as used by plumbers and electricians.

**Soldering wick.** Marketed under the names *Spirig*, *Solder Up* and *Solder Wick*, it's a copper braid impregnated with soldering flux. When heated with the soldering iron it absorbs Solder off the board, thus freeing components. Don't do without this stuff unless you like fried circuit boards and burnt fingers.

**Wirewrap wire.** Also called by the trade name *Kynar*, this is 28- or 30-gauge single-strand wire used to interconnect the pins of wire wrap sockets. It comes in an assortment of colors; get them all, so you can keep data, address, power and ground lines separate.

**Multiconductor cable.** The more flexible wire is easier on the coordination, but also the most expensive. Best buy is *Spectra Twist*, and its kin, from surplus houses. If you need jumper cables, buy them; Making a two-ended, 40-pin jumper cable can be three hours of maddening work.

**Bus wire.** This is solid, uninsulated stuff. A small roll will do for a lifetime. I use it for wiring, securing bulky capacitors to circuit boards, holding bundles of things together and for making special tools.

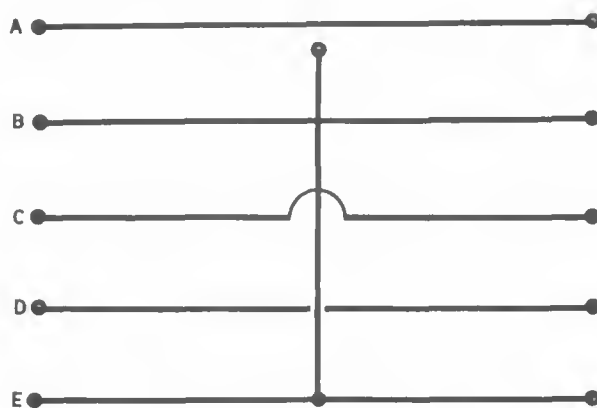
**Miscellaneous.** Sockets, perforated board, mounting hardware, and such will always be needed.

Details about supplies needed for each project in this book will be presented with the project. Except for integrated circuits, most of the items are available right off the shelf at a local Radio Shack or other electronics supply house.

## Schematics

Schematic drawings of electronic circuits are identical to maps. They show routes, direction, junctions, relative importance and functions of locales, two-way and one-way streets, traffic flow and congestion and so forth. At first, the symbols may seem like the mysterious hieroglyphics of a secret society, but their symbolism can soon become as familiar as a roadmap. Even strange places can be assessed from afar.

First, the symbols. A line is a wire running from some point in the circuit to another. Consider the sketches below:

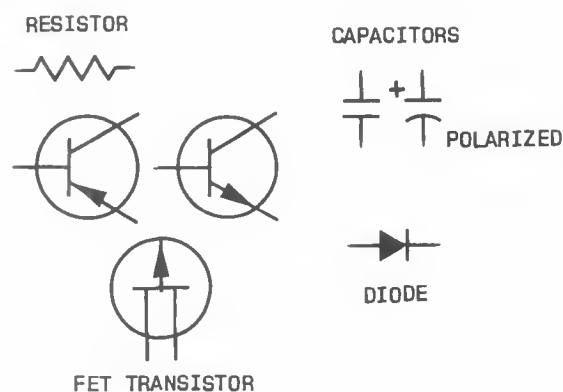


The first drawing is a simple wire. The electrical path moves from one point to another, in either direction. By following the path of a wire through a circuit, the pattern of connections can be discovered. When wires are forced to cross one another, but not to connect with each other, it must be made clear. On a roadmap, non-intersecting roads are shown either by a break in one of the intersecting lines, or in showing interstate highways, merely by crossing one 'below' the other in a different color.

Sketches b, c and d are the three ways of drawing wires which do not connect to each other. The first, simply crossing them, is the most common. The second method places a semicircular bump in the crossing path, and is used by Sams Publications in this country and commonly in Europe. Occasionally the broken path crossing shown in sketch d is used.

When wires connect, a dot is used to clarify that a connection is to be made. Occasionally, you may come across earlier schematics which use the 'bump' method of showing unconnected wires. On these schematics, the lack of a bump indicates wires are connected.

The wires (or patterns of copper etched on circuit boards) connect electronic components. Some of them are:

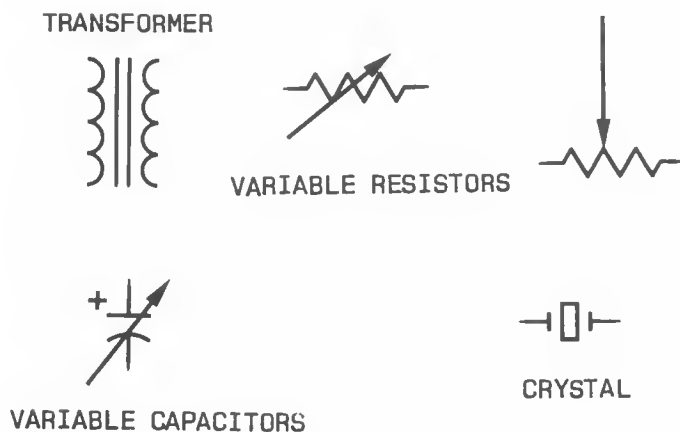


Since this is a lesson in reading schematics and not electronic theory, I recommend that you turn to an excellent book by Forrest Mims, 'Engineer's Notebook', sold by Radio Shack, for an introduction to what each of these parts does. Briefly, the symbol for a resistor has the flavor of a long wire being compressed, meaning the electrical flow is somehow being resisted. The innards of a capacitor generally consist of metal foil separated by a non-conducting paper or plastic, and the capacitor's schematic symbol is fairly representative, with two plates facing each other but not joining.

Some capacitors are designed to fit into a circuit in only one direction; the positive (+) sign identifies that direction. These capacitors are identified on their bodies by a positive or negative sign. Another one direction (polarized) device is the diode. It consists of an arrowhead striking a barrier, implying that current may flow in the direction of the arrowhead, but not back across the plate. The body of a diode may have the diode symbol imprinted on it, or a band to indicate the 'barrier' end.

The transistor usually has three connections (such connections are called 'leads' on small parts such as these). These leads are identified as collector, base and emitter or source, gate and drain, depending on the transistor type. This will be shown on the diagram, and the transistor will be imprinted with the information, or it will be provided on the package in which the transistor is sold.

A few other symbols are:



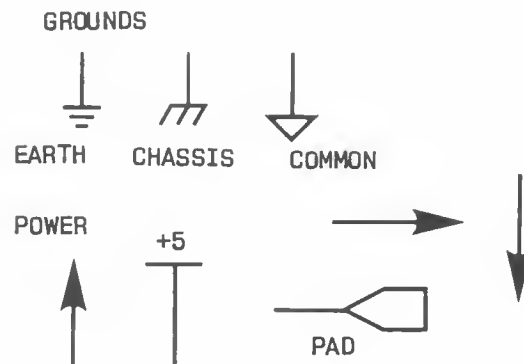
The first is a transformer, whose job it is to take current fed into one coil and induce that current into a second coil. An iron or ferrite center (the parallel lines in the symbol) aids in efficient transfer of that current.

The next three symbols look like resistors and capacitors, which they are. The added arrows show that their values may be varied; hence, they are called variable resistors and variable capacitors. The variable resistor is best known as the volume control on a television, and the variable capacitor is found as the tuning control on a table radio.

The last symbol is a crystal, a piece of cut quartz capable of vibrating (resonating) under certain electrical conditions. Because a crystal is a very accurate, fixed, molecular device, it is capable of resonating (also called oscillating) at

precise intervals. It is used for the master control of all pulses in the TRS-80.

A few directional symbols are now in order:



The first are known as grounds, and they are used to indicate a potential of zero or neutral voltage. The first of the trio is an earth ground, commonly used in radio, television and hi-fi schematics, but purists use it only to describe an actual connection to a ground spike or cold water pipe. The second is a chassis ground, indicating an electrical connection to the metal case which encloses the circuit. It is often (though incorrectly) interchanged with the earth ground.

The last of the three grounds is a 'common' or neutral ground, and the one which is used to indicate the zero voltage line in the computer. All other voltages within the computer system are described in terms of their relation to this ground.

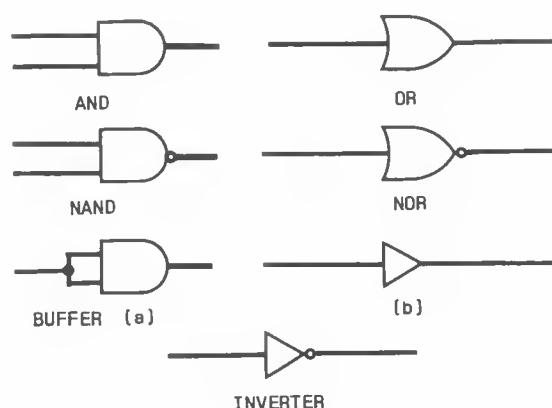
The next quartet of symbols indicate power. The up arrow generally points to an actual voltage value (such as +5 or +12). The horizontal line indicates merely a 'high' level, that is, a connection is made to the normal positive power supply for the circuits in the system (+5 volts in the TRS-80).

Non-positive voltages have no standard symbols. Negative (or below ground) voltages can have either a horizontal arrow or a down arrow, pointing to the voltage desired at that point. The schematic tells you that a connection is made to the voltage level shown.

Another use of a horizontal arrow is to point to important connections to be made elsewhere on the schematic or on other sheets of the schematic. In the former case, the arrow is used because actually drawing the wire may clutter the schematic, making it illegible. When you see an arrow, be sure to find the other end of the connection described (indicating words such as 'clock', 'mem' or 'port FF' may be used as guides to where the connection is made).

Another useful symbol is the last of the group above, the pad. It indicates a significant connection, usually to another device or circuit board. Using this symbol makes it clear that the connection is to be made somewhere off the board on which you are working. In this book, I have not used these symbols where indicating a connection to the TRS-80; instead, the cable to the TRS-80 is shown with the connecting wires striking a wide vertical band marked 'TRS-80 Edge Connector'. Other types of off-board connections, however, are shown with the pads.

The most common families of parts found in computer circuits, however, are shown below:

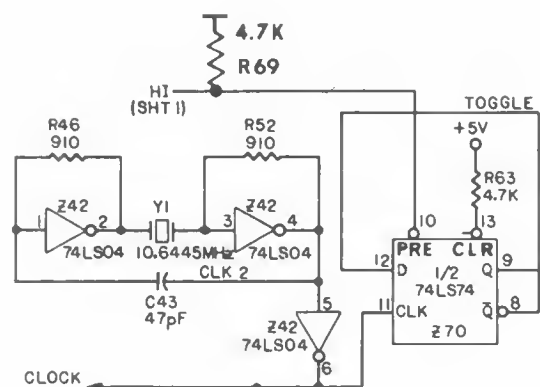


These symbols represent integrated circuits, those multiple lead, buglike packages that handle the bulk of the work in the computer. Briefly, these are logical building blocks. Sometimes there are several blocks in one integrated circuit, and these various blocks may be scattered throughout the circuit diagram. This can be confusing when actually building a circuit, but since pin (lead) numbers are given, you only have to remember where you put the part.

You should know that the TRS-80 *Technical Reference Handbook* uses what are called 'functional' schematic elements, meaning identical parts are not necessarily drawn the same throughout the schematic. I have chosen not to use this method, which, although it makes circuit operation less evident, is clearer when doing actual wiring.

Complete logical and physical diagrams of every circuit used in this book are given in an appendix. Those diagrams will help give you an idea of how these logical blocks are packaged inside 8-, 14-, 16-, 18-, 20-, 24-, 28- and 40-pin cases.

Basically, that covers reading a schematic roadmap. Below is a section of circuit. See how the logic elements are connected to each other as well as to two resistors, a capacitor and a crystal. Notice also that the logic elements are all marked 'Z19', since they are separate blocks within a single component. An arrowhead indicates a wire leading off the board, and power and ground connections are shown. The numbers on the logic elements are the pin numbers for the component connections:



## Be Tolerant

Every electronic component is manufactured to work within specific limits, whether they be accuracy, temperature, speed, power use or other limit. These are the components parameters or *tolerances*. The circuits in this book have been designed to use the most commonly available parts, so the matter of tolerances is rarely important. However, sometimes those tolerances are important, such as when talking about memory speed or power supply voltages.

Power supply should be within five percent of the voltage specified; a supply indicated at five volts may vary only from 4.5 volts to 5.5 volts. By using the power supply regulators shown in the schematics, these voltages should not be of concern. Unless you are familiar with power supply design, do not attempt to use other methods of regulation.

Very few of the resistors have tolerances noted on the schematics. The rule of thumb is one quarter watt at five percent, but if you can only obtain half watt units, or 10 or 20 percent resistors, don't be concerned. The quarter watt

resistors are a bit less costly and are a bit more aesthetically appealing. Consider also that if a resistor is specified as 1,000 ohms, a 20 percent deviation gives a range of 800 ohms to 1,200 ohms. Thus, the standard values of 910 ohms or 1,200 ohms should do as well.

Capacitors are notoriously sloppy in their tolerances, especially electrolytic types (those whose polarity is marked on the schematics). These normally vary from 20 percent low to more than 100 percent high – thus, when a 500 microfarad capacitor is noted, it can range from 400 to 1,000 microfarads. Also, there is some revision in the standard numbering method used for parts values: 470 microfarads is now being called 500 microfarads, for example. So when you try to obtain a capacitor value marked in the parts list, remember that a nearby higher value is fine.

Voltage parameters for polarized (electrolytic) capacitors are important. Never get an electrolytic capacitor with a value less than that specified, but do not hesitate to take one with a higher voltage parameter. That is, a capacitor specified at 47 microfarads, 16 volts, can be replaced with one specified at 50 microfarads, 35 volts. It may be physically larger, but it will work equally well.

If you walk into a store and hand the sales clerk a parts list, don't be surprised if you are asked a few more questions. You might be faced with choosing between parts which are identical as far as the parts list in this book is concerned, but which include other parameters.

Resistors can be carbon composition, carbon film, glass or wire wound. These days, carbon film is common and cheap, and that's your first choice. Carbon composition is the next choice at a lower quality, and glass is excellent but at a higher cost. Forget wire wound, because they can contribute unwanted side effects.

Ordinary capacitors are manufactured in many ways: ceramic, polystyrene, polyester, silver mica, polycarbonate and paper. For the bypass capacitors necessary for all the circuits in this book, ceramic types are your choice. Cheap. If you get silver mica, so much the better, but you'll pay a price. Watch out for polystyrenes or polyesters if you plan to solder, because they are delicate and you can damage them with too much heat. Otherwise they are excellent, but quality overkill. Polycarbonates are slick types, and you might consider using these if you build the 8-track mass storage system. Run the other way if you see paper capacitors.

Electrolytic capacitors come in two basic types – metal cans (covered with plastic), and those manufactured using tantalum (an expensive metal of great strength and purity). For most digital projects, choose the ordinary cans. Tantalums of the same value, although smaller, high quality, and very pretty looking, are costly and not required here.

Digital integrated circuit part numbers are generic, which means that a 74LS00 circuit might be sold as an SN74LS00 or an NEC-74LS00. The prefix characters refer to manufacturers. On the other hand, those parts whose numbers contain 'LS' may not be substituted by parts marked 'S' or 'C' or by those with no markings. 74LS00 may not be replaced by 7400, 74S00, or 74C00, nor may they be exchanged for each other. When integrated circuits are specified, try not to substitute with other circuit 'families'.

This section will not make you a master schematic reader; only practice will do that. Pick up copies of the Engineer's Notebook mentioned above, as well as various of the project books sold by Radio Shack and others.

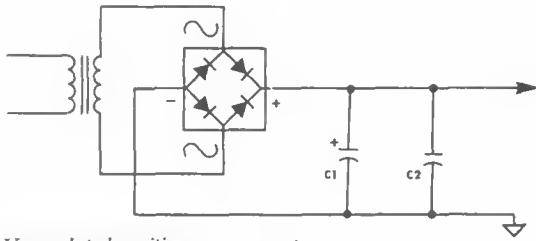
## Building a Power Supply

All the projects presented in this book will either be modifications to the TRS-80, in which case they will draw power from the computer itself, or outboard devices which will need power. There are several ways to get this power:

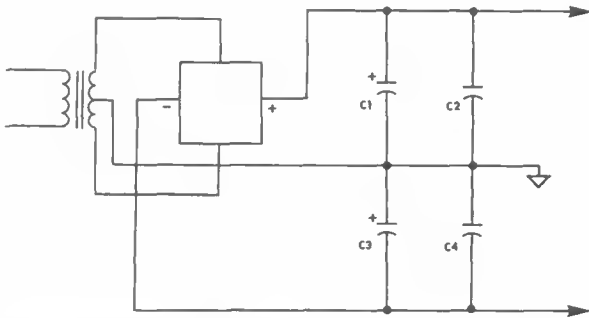
1. Build a power supply from scratch for each device, including transformer, rectifiers, capacitors and regulators.
2. Build a main power supply providing substantial current at between 7.5 and 15 volts, and put a voltage regulator and a capacitor on board each project.
3. Use an assembled power supply providing a 7.5 to 9 volt output for each project, and use a voltage regulator and a capacitor on each board.
4. Use an assembled power supply providing 5 volts and feed all boards from it.

My recommendation? Probably an assembled power supply of 7.5 to 9 volts feeding each project. Power supplies are important to projects, and they should be well regulated, free of residual 60 Hz (Hertz equals 'cycles per second') ripple, and should not tend to transmit signals between projects. Some things to consider include:

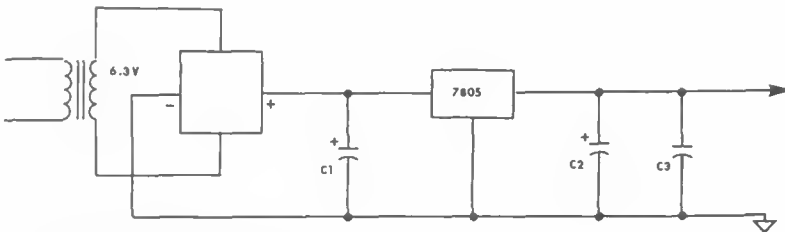
## Building a Power Supply



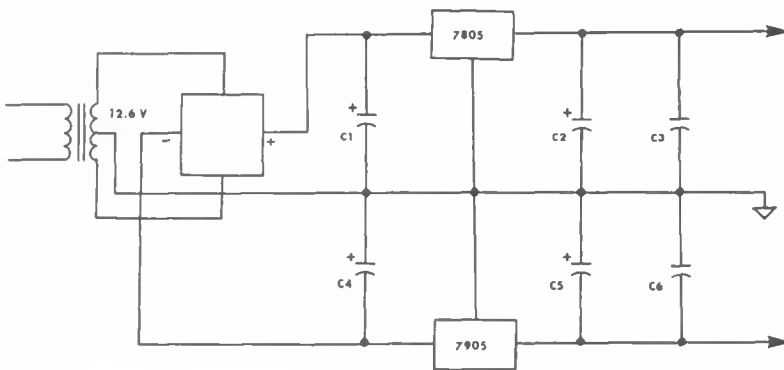
*Unregulated positive power supply.*



*Unregulated bipolar power supply.*



*+5 volt regulated power supply.*



*+5/-5 volt regulated power supply.*

1. Building a power supply is not very time-consuming, but it is probably going to be bulkier than the project being constructed. Then there's the question of where do you put it? If it's in the project, you have to be very careful to shield the AC primary and secondary leads. Also, you have to be sure it's working properly before you can begin to test your project.

2. Small, assembled power supplies are inexpensive. They are normally sold as 'battery eliminators', with their current capabilities specified. The AC leads are encapsulated in plastic with the rest of the supply. Although you need a separate house-current outlet for each of these supplies, the work you do (both building and testing) is lessened and the safety to you and your project is increased.

3. Larger power supplies are expensive to buy and complicated to build. Unless they and each project being fed contain plenty of transient suppression (in other words, lots of extra capacitors), the actions of one device may affect another. But they do tend to be more immune to house current fluctuations than small homemade or purchased power supplies. With regulators on each project, moreover, you can provide more immunity to spikes and fluctuations. They are truly 'brute force' circuits.

4. Large regulated power supplies are highly stable, but expensive. They are capable of feeding a whole range of boards, less house current outlets are needed, and the level of regulation is usually substantial enough to prevent fluctuations at the board level. However, onboard filtering is still necessary to prevent interaction among external devices.

The circuit diagrams that follow present a simple, unregulated, positive power supply; a simple, unregulated, bipolar power supply; a regulated five-volt power supply; a regulated dual-voltage (+5 and +12 volts) power supply; a regulated bipolar (+5 and -5 volts) power supply; a regulated four-voltage (+12, +5, -5 and -12 volts) supply, and last (but by no means least) the design of the power supply (+12, +5, and -5 volts) used in the TRS-80. All these supplies can use the transformer / rectifier 'power supply' sold for the TRS-80 as their source.

## Power Supplies

Digital electronic circuits constructed by hand can perform as well as neatly-laid-out commercial circuits. In some ways, though, these circuits have to be designed better than commercial ones, because professionally etched boards are usually designed with careful consideration given to signal paths. Wire-wrapping or soldering, on the other hand, can look like a rat's nest and sometimes act that way.

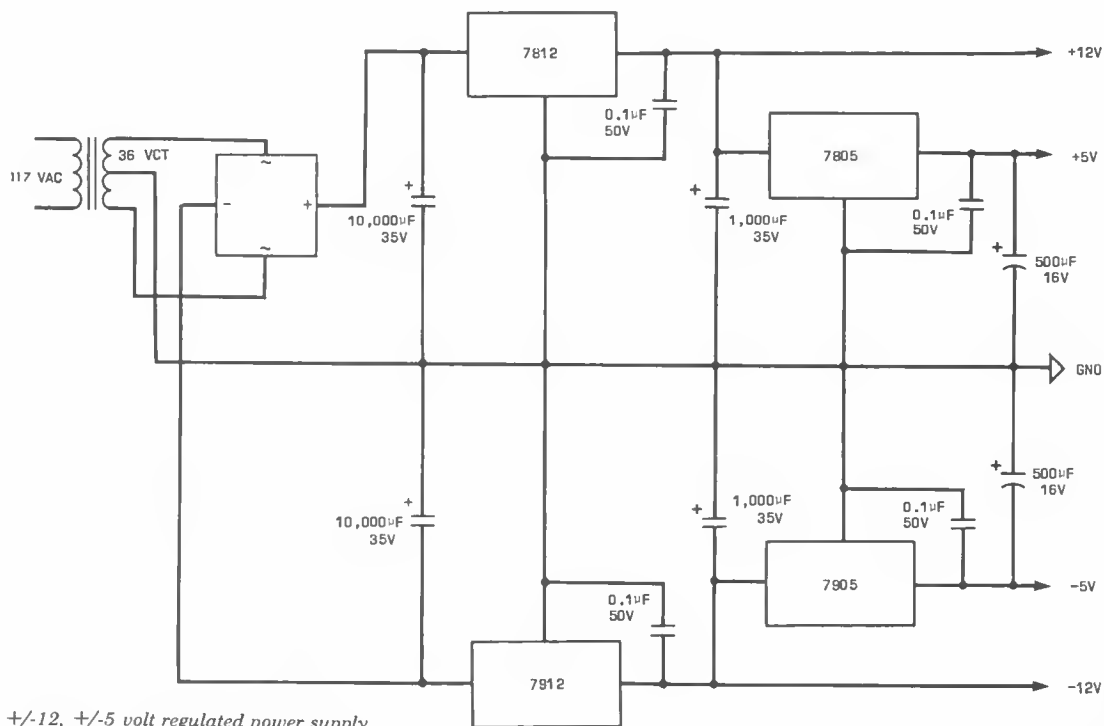
**Most important:** always use bypass capacitors in your projects. These are capacitors with a value of about 100 nanofarads (0.1 microfarads), ceramic discs or dipped mylar, attached between +5 volts and ground. These are *physically* placed near their respective integrated circuits. When I finish wire wrapping a circuit, I give it a quick test to make sure the wires are connected properly and the circuit behaves normally (barring occasional crashes). Then I attach a 100-nanofarad bypass capacitor between the positive power supply and ground, directly between the wire-wrap pins carrying the power.

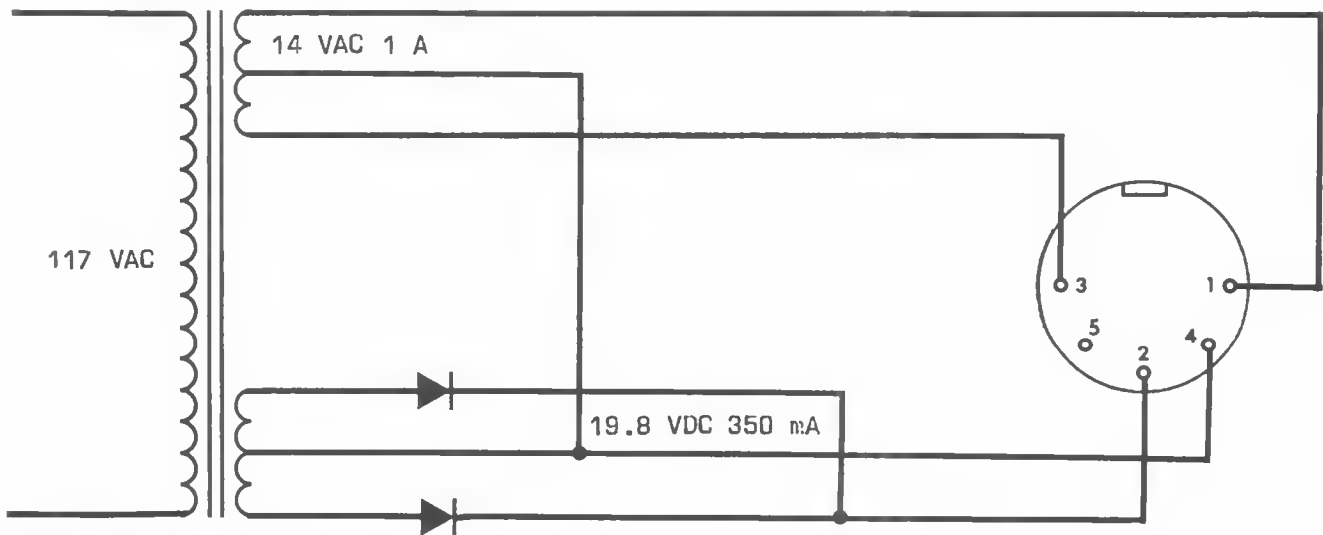
Secondly: add termination resistors to the data lines whenever more than one project is connected to the computer. Termination resistors act as electronic clamps, holding the

lines steady as the signals sweep through. These eliminate random noise, as well as signal 'overshoot', caused by capacitance introduced by the wires themselves. If you have an expansion interface of the new breed (no buffered cable), termination resistors are already in place. They are also standard with the *LNW* and *Microtek* interfaces (though I recommend doubling the value suggested by *LNW*, to at least 470 ohms and 1,000 ohms).

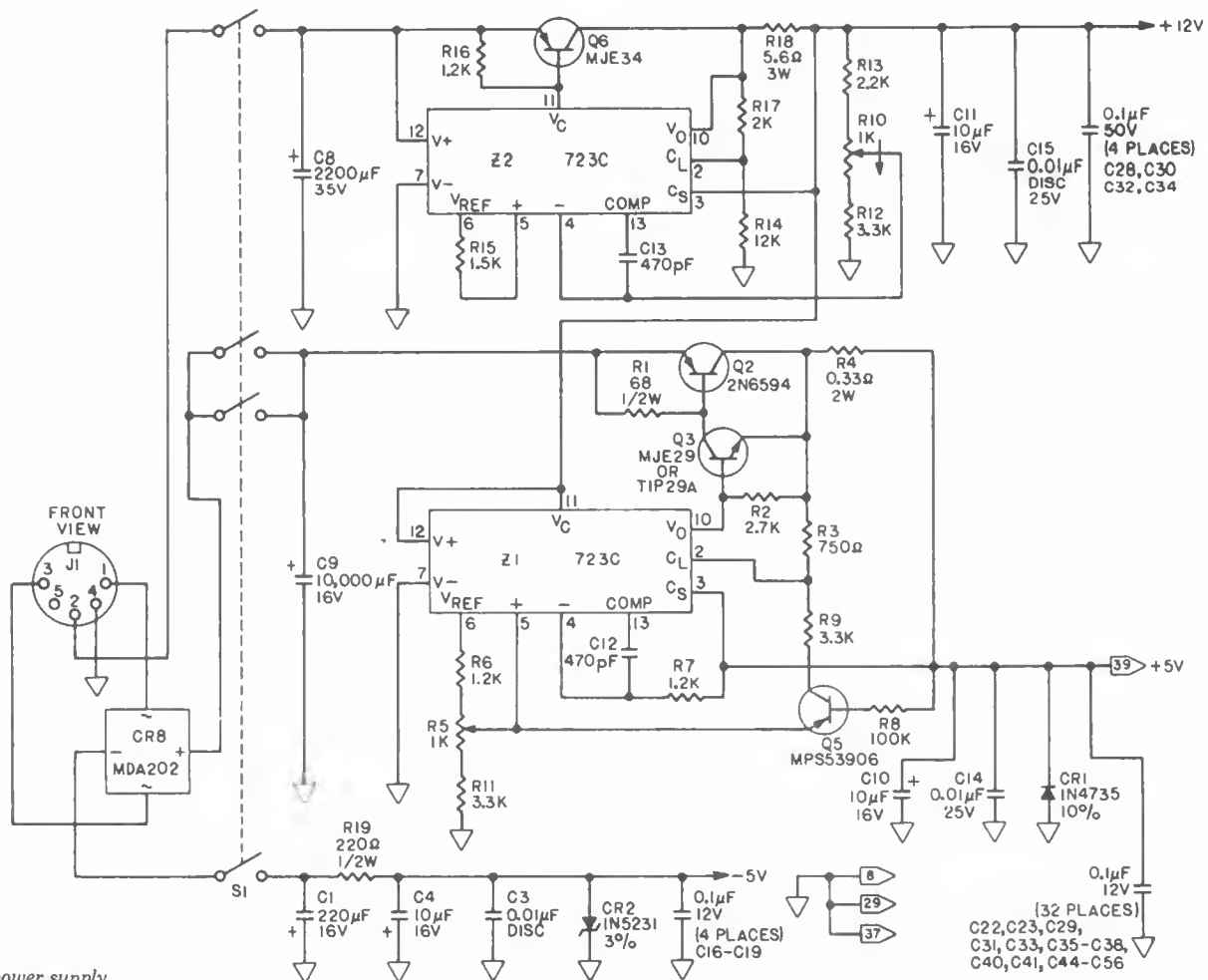
The simplest addition of termination resistors involves connecting a 1,000 ohm resistor from each data line to ground. To ensure even better signal clarity (and also to maintain the 'high' level the computer normally sees on its lines), also attach a 470 ohm resistor from each data line to +5 volts. In all, then, eight resistors will be needed for each project you add.

Finally, do not skimp on the power supply. Generally you will see a 2,200-microfarad capacitor followed by a regulator, a 470-microfarad capacitor and a 100-nanofarad capacitor. This is the absolute minimum configuration for a working power supply. If you have the room, increase the value of the first capacitor to 4,700 or 10,000 microfarads, and place a 100-nanofarad capacitor between input and ground physically near the voltage regulator.





TRS-80 transformer adaptor and supply.



TRS-80 power supply.

### Those Colors: What They Mean and How to Read Them

The color codes used for resistors, capacitors, and other parts are brought to you by the same folks that brought you phrases like 10W-40 and RS-232C: the standards-setting powers of the engineering industry. It becomes an international shorthand.

The colors are black, brown, red, orange, yellow, green, blue, purple, grey and white. If you can't immediately remember that, then pick up a piece of multi-conductor 'rainbow' cable. The colors are all there in the same order. The table below presents the color codes and how they can be read on the bodies of resistors, capacitors, and diodes.

FIRST AND SECOND COLOR BANDS		THIRD COLOR BAND	
BLACK	0	BLACK	0
BROWN	1	BROWN	X 10
RED	2	RED	X 100
ORANGE	3	ORANGE	X 1000
YELLOW	4	YELLOW	X 10,000
GREEN	5	GREEN	X 100,000
BLUE	6	BLUE	X 1,000,000
VIOLET	7	SILVER	100
GRAY	8	GOLD	10
WHITE	9		

FOURTH COLOR BAND IS THE TOLERANCE		
GOLD = 5%	SILVER = 10%	NONE = 20%



What do these values mean? Resistance is a kind of objection to electron flow, measured in ohms (pronounced with a long O). The abbreviation is a Greek omega ( $\Omega$ ). Thousands of ohms are kilohms, or just kilohms, and abbreviated K (k in Europe). Millions of ohms are megohms, abbreviated simple M. The ability of a resistor to withstand electrical current is measured in Watts (W). Most

computer work is done with 1/4 Watt resistors.

For resistors without color bands, the values are stamped on using R (instead of omega) for ohms, K and M.

Capacitance is the inclination of a non-conducting object to store an electrical charge, measured in Farads. The abbreviation is a capital F. Since this is a very large amount of capacitance, real work is generally done in millionths of Farads, or microfarads (mF), and millionths of millionths of Farads, called picofarads (pF). Since many of the more popular capacitance ranges for computer work fall between these two figures, the abbreviation for thousandths of millionths of Farads, or nanofarads (nF) is common in Europe. The ability of a capacitor to withstand voltage is measured in voltage tolerance (V).

Capacitance is usually printed on the capacitor in mF; color bands are rare. Picofarads are marked 'p'; the absence of an abbreviation indicates microfarads. Note that these capacitor 'base values' are equivalent: 18=20, 27=30, 39=40, 47=50.

-----  
 Abbreviations and Conventions Used  
 in this Book  
 -----

AD-A15	Computer address lines.
C	A capacitor, specified in the parts list; positive end negative side as marked.
CLK	Clock, usually a flip-flop input.
CLR	Clear, usually to flip-flop or latch.
CR	A diode, Radio Shack schematic reference.
D	A diode, specified in the parts list.
DO-07	Computer data lines.
F	Farad; used only as part of mF, uF, pF.
GND	Common computer ground.
K	A relay, specified in the parts list.
-K	Resistance value suffix = x 1000.
LED	A light-emitting diode, specified in the parts list.
m-	Capacitance prefix for x 0.000 001.
M	A motor, specified in the parts list.
-M	Resistance value suffix = x 1,000,000.
MHz	Megahertz (million cycles per second).
p-	Capacitance prefix = x 0.000 000 000 001
P	Piggybacked integrated circuit.
PRE	Preset, usually to flip-flop or latch.
Q	Transistor, specified in the parts list.
Q	Flip-flop or latch output.
R	A resistor, specified in the parts list.
S	A switch, specified in the parts list.
T	Transformer, specified in the parts list.
u-	Capacitance prefix for x 0.000 001.
U	Integrated circuit (LNU references only).
Vcc	Collector-supply voltage, that is, the 'five-volt supply'.
W	Watt; resistor power reference.
X	OIP shunt or jumper, R/Shack reference.
X	Crystal oscillator.
Y	Demultiplexer output signal.
Z	An integrated circuit, specified in the parts list.
*	Used in text references for 'not', in place of a horizontal bar across the top of the abbreviation.
0V	Ground, zero-level, or center-ground.
+5V	Five-volt positive regulated power supply, the 'Five volt supply'.
-5V	5-volt negative regulated power supply.
+12V	12-volt positive regulated power supply.
117V	House current, 105-120 volts AC.

BASIC commands; Z-80 mnemonics and opcodes; CPU and other schematic signals are printed in UPPER CASE letters. Hexadecimal numbering is printed in **BOLD** letters.

---

---

# 1

---

## Getting Inside

A TRS-80 is the proverbial black box. It's the first mass-marketed personal computer that was intended for simple home and business use. As far as retail sales are concerned – whether it's a car, a hi fi, or instant pudding – thinking of it as a black box is just fine. “Yes, it's a complete computer with a keyboard unit, video display, and cassette storage.”

To have this small computer work to even a fraction of its potential, though, you've got to have the power to control it. As with driving a fine automobile, that power comes with understanding, comes with being able to look through and into the box, comes with keeping it carefully tuned and customized so that it can respond to your wishes.

In this Chapter we'll open the electronic black box a little at a time. If this is all new to you, prepare for some exciting finds inside this box; if not, then follow along and discover just how much you've learned about microprocessors in their own age.

### What You See

The main computer unit is in a small case looking something like a typewriter that has lost its carriage. There are 53 keys in a typewriter style layout, and perhaps on your unit a numeric keypad to the right. Connections on the back are marked *Power*, *Video*, and *Tape*, along with an on/off switch.

The unit is entirely silent. It is electronic in conception and execution.

A power supply lowers and isolates the 120-volt household supply to approximately one-seventh that value, and that feeds the

keyboard unit. An ordinary cassette player is cabled in, and a partly disemboweled television set plugs in place, serving as a crude video monitor.

The designers of the TRS-80 struggled with, and won, the battle of familiarity. Televisions, cassette recorders, and typewriters are among the most ordinary of home or office appliances. But by winning this battle of familiarity, the designers also clearly set themselves up to lose the battle of reliability (more on that later).

Let's first take a look at some of the things the computer does, and then very generally try to discover how and with what the machine does them.

When you power up your computer, you expect to be able to communicate with it. Unlike a television, it does not entertain, but rather evaluates and responds with an electronic psyche. If it were not capable of using a human-like language, we would be forced to use the machine's language. But since it will be asked to accomplish human tasks, we will demand that it speak a human sort of language – BASIC (Beginners' All-purpose Symbolic Instruction Code). BASIC has grown since its humble but inspired beginnings at Dartmouth College into a formidable tool capable of rocking other standard languages off their computational pedestals. If you have been using your TRS-80's BASIC, you know its fluidity. But it does not work alone.

The BASIC that is in the TRS-80 works hand-in-hand with the electronics to produce a video display for us to read, and to examine a keyboard for us to type on. The keyboard gives

input to the computer, the video display shows output from the computer. Input and output are grouped together in computer terms, and are collectively called *I/O*. The tape recorder, which saves and loads computer programs, would also be called *I/O*.

Forget the claims that the TRS-80 computer can do what this-or-that state-of-the-art computer could do five years ago. Maybe so, but it really can't. Because it isn't built like a piece of office furniture. Because it doesn't act, work, or 'think' the same. No matter how you paint it or pad it, a four cylinder sportscar will never be a luxury V-8 sedan. And Burger King is not a candidate for the four-star list. But they can do *what* they can do as well or better because of the simple, direct, streamlined nature of their operation and conception. Turn away the question of value judgment, and you discover that – considering portability, cost, ease of maintenance, accessibility and vastness of the software domain – the TRS-80 is probably a dimension *better* than that recently demised state-of-the-art dinosaur.

That about covers what we can say about the minimal TRS-80 setup – a keyboard computer unit with *I/O*. Before popping off the cover, let's name some of the rest of the *I/O* devices that can be hooked up to the machine. Those might include:

### **An Expansion Box.**

This attachment expands not only the internal capacity of the computer, but also forms an electronic saddle, permitting other devices to ride on the back of the keyboard unit.

### **A Printer.**

Obviously, an attachment to provide a permanent record of the machinations of the TRS-80.

### **A Diskette Drive.**

A place to store information and programs when the computer is turned off; it is very like the cassette player, except that it is speedier and can be accessed differently. More on that later.

### **Voice Input/Output.**

An ability to speak and be spoken to on the part of the computer. This is one of the experimental options.

### **Telephone Communications.**

Communications with other computers,

similar or different, nearby or across the world.

### **Control Centers.**

The power to change, activate or extinguish electrically-powered equipment throughout a home or office.

There are more devices which might be considered, including clocks to tell the time of day, little circuits to create sound effects and music, even other computers used as 'slaves'. In fact, where electrically controlled equipment is involved, almost anything can be attached directly or indirectly to the TRS-80.

## Hesitation

When you open the computer's case, you'll see an enormous amount of electronic circuitry. Before you open it though, you should have an idea why there are so many *integrated circuits* and circuit board *traces* connecting them.

We live in an analog world. We judge size or volume or loudness not by how big or full or loud it *is*, but by how big or full or loud it seems in relation to something else, even if that 'something else' is merely what we are used to hearing in our normal world.

In other words, all our evaluations are made by analogy. "How big is it?" "As big as a basketball!" "Is she pretty?" "Pretty as a picture!" "Is it far?" "About a stone's throw from the corner." Our cliches are built on comparison or analogy. Ideally, then, we might like to build a machine that work for us in our own terms . . .

"Machine!"

"Duh, yessir, sir."

"Add fourteen and thirty-seven, machine!"

"Yup, yup. Lessee. Hmmm, fourteen is **this big**. And thirty-seven is **this big**. That makes a number **this big**. . . . Sir?"

"Yes, machine?"

"That's as much as thirty-seven and fourteen, sir! Looks just like fifty-one, sir."

"Thank you, machine."

This computer has a rather limited voice capacity, but what it did inside itself was electronically quite sophisticated. It took in a value and transformed it into an electrical voltage of fourteen units, stored it, accepted a

second value and transformed it into a voltage of thirty-seven units. Then it added the voltages together; the resulting voltage worked out to be equivalent to a value the size of fifty-one voltage units.

The manufacturers of the real world might be able to create electronic parts with this kind of accuracy, but chances are that these parts wouldn't be cheap. A TRS-80 made with them would be worth more than the previously mentioned V-8 auto.

So in the stone age of computer activity, it was decided that the simplest level of evaluations would have to be made. A low voltage or 'off' condition would be made equivalent to zero; a high voltage or 'on' condition would be set equal to one. That was it. All calculations would have to be done with ones and zeros – the binary system.

You've probably heard of, and perhaps used, binary numbers and there will be more on this system later in the book. But the point is that you can well imagine that doing work with real, human numbers means quite a basketful of the individual ones and zeros. Which means, consequently, many separate signal lines to carry those groups of numbers.

## So Open it Already!

Turn the power off to your TRS-80 keyboard unit and flip over on its front. Carefully undo the cabinet, noting the different sizes of screws used to fasten it together. Holding the entire computer firmly together, again flip it on its back. Remove the top cover. Gently lift the keyboard forward and remove the five or six plastic spacers underneath it; in later 80's, one of these will be solid and the rest flexible. Note their positions. Now set the keyboard back, lift the electronics out of the case and place it on a spacious work surface.

Time out. As you make changes to your TRS-80, add memory and the like, you will be opening the case many times. Two things might happen. First, the keyboard grommets will get lost. Their purpose is twofold: to prevent the keyboard from shorting against the main circuit card, and to cushion delicate parts against a constant onslaught of none too gentle typing. If you lose them, cut new ones out of bottle corks. They'll work just fine.

Another more difficult problem is the keyboard cable itself. This is a band of springy copper leads covered with white plastic located

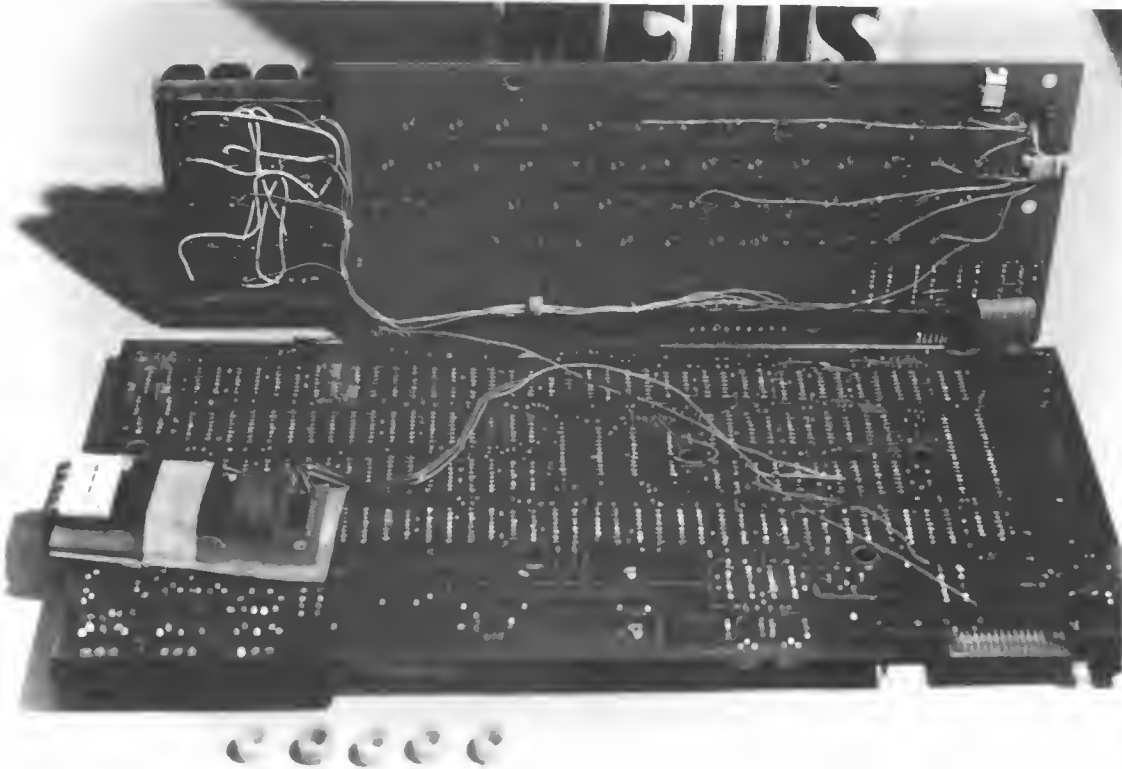


Photo 1-1. The TRS-80 opened and spread out.

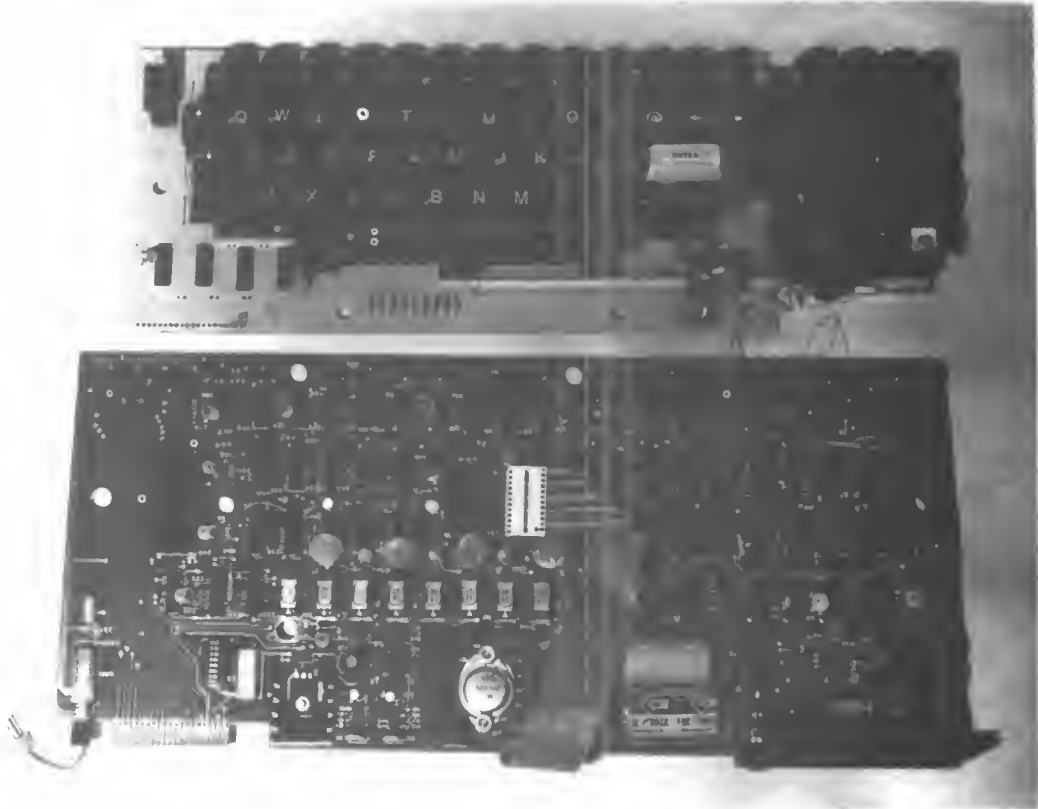


Photo 1-2. Closeup of keyboard cable area.

*Complete TRS-80 spread out for cable replacement. Other visible changes include Level I/II switch on keyboard (bottom right of keyboard), new Level II interconnect cable (center), additional keyboard socket (top left), and external reset switch connector (bottom left).*

at the bottom left of the keyboard. It electrically, and physically, attaches the keyboard to the main circuit board. Very subtle cracks can occur in the copper after about a half-dozen flexes. Symptoms can be lost letters or odd combinations of letters, a system constantly crashing or otherwise acting up, constantly repeating letters, or complete lockup of the keyboard. *Avoid flexing this cable quickly or bending it sharply.* Later we'll replace this cable with something better, but for the moment be gentle with it.

Now back to the machine. Spread the two sections connected by the keyboard cable out in front of you. The keyboard and the top of the circuit card will now be visible. The Central Processing Unit is a single integrated circuit, the Z-80; it is to the far left in the photograph on this page. The power supply is the block of 'heavy equipment' at the back; the two small potentiometers regulate the voltage to within five percent, so keep clear of these controls. The eight memory circuits are in a row near the power section, and the language memory (Level I and

newer Level II BASIC) is located in the center of the circuit card. On most Level II units, this language required three integrated circuits, and so these were placed on a separate two-by-three-inch card taped to the main board and connected with a 24-conductor cable. *Don't be tempted to remove that cable, yet!*

A group of circuits to the bottom left in the photograph hold the video image. Two important parts are socketed on the board; these are marked Z3 and Z71, and they are programmable shorting jumpers. What makes them programmable is the fact that with a gouging tool you can break the connections; not very subtle, but it works. Their purposes are different – one selects the amount of memory available in the keyboard unit, and the other selects which of the standard languages (Level I or Level II BASIC) is in use.

At the top left, the video and cassette output is controlled. Two small potentiometers on that side of the board position the video image on the screen, so if you received a TRS with its image off center, twiddling these will straighten it out. The

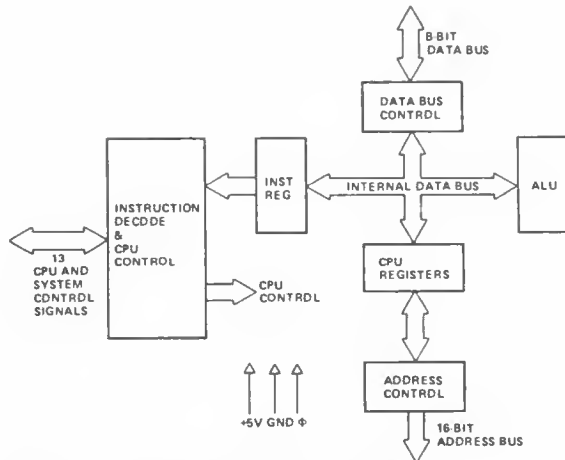


Figure 1-1. Z-80 block diagram from Zilog.

remainder of the board is dedicated to controlling the complex electronic traffic.

One last item is the edge card connector, which can be viewed at the top right hand side of the photograph. Through this connector, the TRS-80 may speak to the outside world. Much more on that connector later in this book.

On the keyboard itself there are a few circuits. Their job is to assist the Central Processing Unit (CPU) in determining which key is pressed. These circuits are sometimes remarkably sensitive; there will be more about these later.

## Put it Back

Become familiar with the layout of this board, as it will assist you in locating work or repair areas. Functions of the computer parts are generally grouped together, so modifications will usually be restricted to a compact area of the board. Once you have a good idea of where everything is, put the keyboard unit back together carefully.

To really get to understand your machine you will want to obtain, either off the shelf at Radio Shack, from a Repair Center, or 'National Parts', the following manuals:

*TRS-80 Micro Computer Technical Reference Handbook.* Catalog No. 26-2103.

*TRSDOS & Disk BASIC Reference Manual.* Catalog No. 26-2104.

Printer Cable Service/Installation Manual (Order).

Expansion Interface Service Manual (Order).  
Make sure this includes the FD1771 disk controller appendix from Western Digital.

16K RAM Expansion Service Manual and Addenda (Order).

*TTL Databook* (National Semiconductor). Catalog No. 62-1370. This may not be stocked anymore; if so you can order it from National Semiconductor direct.

And finally, a *Z80-CPU / Z80A-CPU Technical Manual* should be ordered from Zilog, Inc., 10340 Bubb Road, Cupertino, CA 95014. It costs \$7.50.

These references will not only help you to make the modifications suggested in this book, but also to understand the operation of the computer, bend it to your wishes, and repair it if it fails. Additional references which you will find valuable are listed in Appendix II.

## The Hidden Insides

It's not much of a mystery to TRS-80 users that all that hardware is controlled by software. That's one of the first things you learn. But it's also as simplistic as saying that the driver makes the car go, and just as misleading. Complete computers are called 'turnkey' systems because they imply simple, appliance-level setup and use. But a customized TRS-80 suggests something more, and with that 'something more' comes a requirement to have a better handle on hardware and software.

Let's define some terms. The TRS-80 is a personal computer, which is a popular way of saying a small, microprocessor-controlled, turnkey computer. The microprocessor, a Z-80, is a complicated, general-purpose, electronic switching center capable of accepting, changing, sending, and re-routing a complex array of electronic signals.

Both the TRS-80 as a whole, and the Z-80 in microcosm, have something called *architecture*. Architecture is the overall dimension by which these devices define their electronic space. Put simply, it is 'how they work'. As with all human work, this involves defining tasks and their order, executing those tasks, and producing a result.

In the Z-80, the architecture involves accepting electronic signal groups called *instructions*, decoding them into internal activities, and executing those activities. An arithmetic logic unit (ALU) performs simple mathematical functions, internal memory cells called *registers* hold signal information to be acted upon, and an internal *bus* controls the flow of electronic traffic. The order of entering signals is identified by means of an *address*, which

identifies a fixed numbered slot in the Z-80's electronic universe.

In the block diagram of this activity, shown opposite, note the terms '8-bit data bus' and '16-bit address bus'. The Z-80 is an integrated circuit with 40 external connections. The number 40 is arbitrary, chosen because manufacturing precision is currently limited to a physical 'package' of that size. That precision is also central to why the binary system is used, as mentioned earlier.

From the viewpoint of ten-fingered humans, it would be simpler to do our computing in familiar decimal form. As with the uninspired conversation between human and computer presented earlier, different levels of voltage could be used. But such levels of precision are difficult to produce commercially and impossible to diagnose when they fail.

That is the practical origin of the simple one/zero, on/off, true-false system of numbering used by the computer. To discover how these signals are arranged, we now turn back to the Z-80 itself and the 40 external pin connections. Information put into and called for out of the processor is called *data*, with a simple small value of zero. Any reasonable number of pins could have been dedicated to accepting data, but, based on the amount of work the processor had to do, eight of the 40 pins were assigned. This is the 8-bit (binary digit) data bus.

Likewise, there is the need to determine the order in which the processor is to do its work. In order to move from command to command, it identifies the order of those commands by their position, or *address*. Sixteen of the Z-80's pins are used to specify a number from 0000 0000 0000 0000 to 1111 1111 1111 1111.

When the power is turned on, this *address bus* switches to all zeros. The computer fetches its first command along the *data bus* from that all-zero location. It executes the instruction, fetches the next, and the next, and the next.

There's certainly more to this, but very basically that's the architecture of a microprocessor. Address lines for locations and order, and data lines for information. Now we'll turn to the TRS-80 and its architecture.

The TRS-80 contains the microprocessor, which is the Central Processing Unit (CPU) of the computer. Along its address bus is found the computer's internal information and instruction storage block – its *memory*. Also along this bus will be found video memory, a block which is reserved to display information to us on the screen; a keyboard, which is given its own set of addresses; the BASIC language; some unused areas; and single memory slots in which are housed windows to the cassette recorder, disk drives, printer and so forth.

This hardware is already familiar, so let's look at the hidden insides, the software. The BASIC system is found in permanent, Read-Only Memory (ROM). It consists of several major sections:

- A keyboard scanning routine to discover and interpret activities at the keyboard.
- A video processing routine which presents and updates the monitor information.
- Input/output controls for saving and loading program material and operating a printer.
- An interpreter capable of transforming the 'English' words which make up the BASIC we know, and determining what computational actions should be taken.
- Memory-management systems which apportion the computer's available memory into blocks which will not conflict.
- Arithmetic- and text-processing subroutines which can perform calculations and operations on numbers and alphanumeric characters.

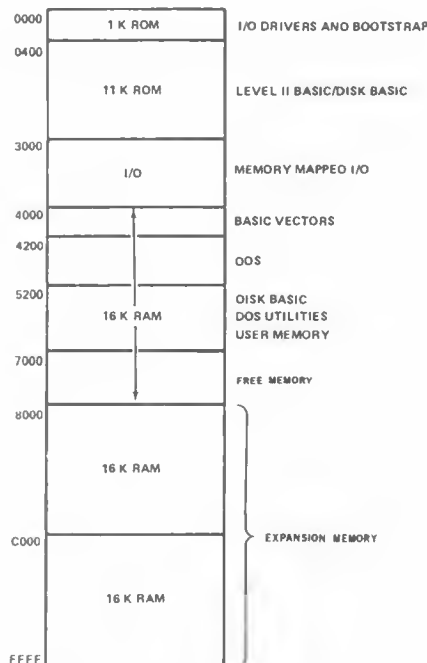


Figure 1-2. TRS-80 memory map.

Overall, the Level II BASIC language requires more than 12,000 separate 8-bit groups, known as *bytes*, to perform its work.

At the end of this Chapter is a detailed look at how the software operates from the time the machine is switched on to the time you read **MEMORY SIZE?** on the screen. In summary, that software disables any signals which might interrupt its operation, turns off the cassette relay and clears data from that output, restores the video screen to 64 characters per line, and sets up a block of memory for use by BASIC programs. A disk drive is searched for, and if one is found, a group of procedures are initiated in order that the disk program may take control of the TRS-80.

If no disk drive is found, the screen is cleared with blanks, the **MEMORY SIZE?** prompt is displayed, and a keyboard scanning process is begun. A valid response to that question is accepted, and, if necessary the entire bank of memory is tested. Error messages are updated on the video screen as needed. Finally, the memory size and available room for text strings is created, the **READY** prompt is displayed, and control of the TRS-80 is given to the user.

## Power-Up Routines

The initialization routine of the TRS-80 is a complicated and very interesting aspect of the computer. It must, of course, set up all the parameters that will be used by BASIC programs, but it also conducts a series of tests and makes hardware adjustments to the device.

It double checks to assure the proper operation of memory, and to be certain that the parameters needed for proper operation of programs will be present. This section will take a look at the initialization process.

Here are the first few instructions:

0000		ORG	0000H
0000	F3	OI	
0001	AF	XOR	A
0002	C3 74 06	JP	0674H

At power-up, the Z-80 chip 'homes in' on address 0000, and begins its execution there. The first action is significant: DI (Disable Interrupt) keeps the clock 'heartbeat', generated by the expansion interface, from disturbing any actions of the computer – especially important, since the necessary software for handling that interrupt request is not in ROM, but rather a part of the disk BASIC, or what is also offered as 'Level III' BASIC.

So the interrupt is masked out. XOR A is the process of 'exclusive-ORing' the accumulator.

Exclusive OR is a logical operation which states: of two elements, either may be zero or one, but not both. Thus, whatever is present in the accumulator is XORed with itself. Since each bit is identical with its exclusive-OR partner, each bit will be set to zero. This effectively clears the accumulator, readying it for processes to come.

The final instruction of the group, JP 0674, gets out of the way of the Z-80's low memory, for it is in this area that the chip's restart codes – very frequently used subroutines – are found. Going on:

0674	03 FF	OUT	(0FFH),A
0676	21 02 06	LO	HL,0602H
0678	11 00 40	LO	DE,4000H
067C	01 36 00	LO	BC,0036H
067F	E0 80	LDIR	
0681	30	DEC	A
0682	30	DEC	A
0683	20 F1	JR	NZ,0676H

After the jump to 0674, the routine resets the output flip-flop at port FF (255 decimal). This flip-flop controls both cassette functions and the 32/64 character video, and by outputting the value in A (0, since it was exclusive-ORed earlier), the cassette will be off, no data will be present at its input, and video will come up normally.

Following this is an interesting (and encouraging) piece of code. Using the Z-80's powerful block move command LDIR, 54 bytes stored at address 06D2 are transferred to the RAM address area starting at 4000. These are the most important pieces of information the TRS-80 must have, so the writers of this program took great care to assure that this transfer would be certain. The LDIR instruction itself takes data stored at an address specified by register pair HL (in this case, 06D2), and moves a block whose length is specified by register pair BC (36 hex, or 54 decimal), to the location indicated by register pair DE (4000).

The interesting part is found just below. The value in A (0) is decremented twice (to FE), and the identical transfer instruction is repeated. This goes on until it reaches zero again, a total of 128 times! We may draw the conclusion that the Z-80 chip probably reaches full power and begins operating before memory gets to the point where it can reliably accept data . . . therefore, the instruction is repeated for a period of approximately 15 milliseconds.

Now a portion of RAM is cleared to zero with the following few commands:

0685	06 27	LO	B,27H
0687	12	LO	(0E),A
0688	13	INC	DE
0689	10 FC	0JNZ	0687H

Recall that after the previous setup process, the accumulator again contains zero. Here a block of RAM specified by the DE register (essentially where we left off) is loaded with zero. Using the fast DJNZ (decrement B, branch if not zero) instruction, 39 bytes are fixed at zero.

A few instructions follow that are very significant at power-up. Address 3840 contains the keyboard row where the BREAK key sits. It is connected to data line 4; thus the instruction AND 04 checks to see if it is held down. If it is not being held down, the result of the AND instruction will not be zero . . . and a jump to address 0075 will be made. This is why expansion interface owners without disks must press the break key at power-up:

06BB	3A 40 3B	LD	A,(3840H)
0BBE	E6 04	AND	04
06B0	C2 75 00	JP	NZ,0075H

How does the TRS-80 find out that a disk drive is in fact connected to the interface? The answer to that – and the reason that the computer ‘hangs up’ when an expansion interface is connected without a disk – is found in the next few bytes of code:

06B3	31 7D 40	LD	SP,407DH
06B6	3A EC 37	LD	A,(37ECH)
0699	3C	INC	A
069A	FE 02	CP	02
06BC	0A 75 00	JP	C,0075H

The stack pointer is set at 407D for use by potential future programs; it is out of the way of all the BASIC pointers set up in the first data transfer, an obvious but important action.

The accumulator is then loaded with the contents of memory location 37EC. There is no ‘memory’ per se at address 37EC; it is instead an instance of ‘memory mapping’. That is, when this memory cell is read, a signal is sent to the expansion interface. This signal strobes information from the Floppy Disk Controller (FDC) to the TRS-80. What will it find?

If no expansion interface is connected, there is no signal to strobe. Hence, the value will be floating, not pulled to ground (zero) on any bit. The computer sees all bits apparently ‘high’ at this location, and interprets it as binary 1111 1111, that is, hex FF.

The next instruction increments the accumulator, in this case resulting in FF plus 1, or 00. In the following instruction this value is compared to 2. A compare (in effect a subtraction, with no ‘result’) will cause the carry flag to be set, since 0 minus 2 is negative. (Note:

Why compare with 02? Why not just 01, as a carry would still be generated? My suspicion is that it is possible for those data lines to ‘float’ in the low state. In that case the CPU would ‘see’ 0000 0000, with the INC A instruction resulting in a value of 01 – which is still incorrect. So a compare with 02 guarantees the presence or absence of the disk controller.)

Once the carry flag has been set, the instruction JP C,0075 would be executed, sending the program to address 0075. For the moment, however, let’s assume that an expansion interface is connected to the TRS-80.

The FDC, when queried by the command LD A,(37ECH), will respond with 80. Incremented by one it becomes 81, and comparing it with 02 generates no carry. The JP C,0075H is thus ignored, and the program simply goes on to find:

069F	3E 01	LD	A,1
06A1	32 E1 37	LD	(37E1H),A
06A4	21 EC 37	LD	HL,37ECH
06A7	11 EF 37	LD	DE,37EFH
06AA	3B 03	LD	(HL),3

The accumulator is set to 1, and address 37E1 is made to accept the contents of the accumulator. Again, 37E1 is a location memory mapped in the expansion interface. This code starts the disk drives rotating (or keeps them rotating), and selects drive zero.

That done, it loads HL with the disk controller address (37EC) and sends out a ‘restore’ command, which tells the controller to move to track zero. Thus, the data will come from drive zero and track zero. Register DE is prepared by loading it with the disk’s data address, 37EF. Now:

06AC	01 00 00	LD	BC,0000
06AF	C0 60 00	CALL	0060H
0060		ORG	0060H
0060	0B	DEC	BC
0061	7B	LD	A,B
0062	B1	OR	C
0063	20 FB	JR	NZ,0060H
0065	C9	RET	

This is a short but very useful subroutine. You may in fact want to call this yourself from time to time. Found at address 0060 is a simple delay loop; load the BC register pair (as is done just before the CALL instruction), and it is decremented and tested until it reaches zero. When it finally reaches zero, a return instruction sends the Z-80 back to the main program flow.

Why a delay? Merely to give the disk drive time to come up to speed, obvious but very

important. Moving ahead with this branch of the program:

```

06B2          ORG      06B2H
06B2      CB 46      BIT      0,(HL)
06B4      20 FC      JR      NZ,06B2H

```

This is a loop which waits until the disk control chip says, "Okay, disk is up to speed and everything looks pretty good", and sends along a zero. The program loop tests this bit until it receives a zero. It is this loop which is maddening to you expansion interface owners who have no disk drive. Like all the previous memory-mapped addresses, 37EC will never have that zero. If a disk is present and all is well, the loop will have found the acknowledging zero sent by the FDC:

```

06B6      AF          XOR      A
06B7      32 EE 37    LD      (37EEH),A
06B8      01 00 42    LD      BC,4200H
06B9      3E BC      LD      A,BCH
06BA      77          LD      (HL),A
06BB      CB 4E      BIT      1,(HL)
06BC      2B FC      JR      NZ,06C0H
06BD      1A          LD      A,(DE)
06BE      02          LD      (BC),A
06BF      0C          INC      C
06C0      20 F7      JR      NZ,06C0H

```

The accumulator is cleared again, and the BC register is set to 4200. This will be an area of RAM set aside for disk use. 37EE is loaded with 0, and 37EC is loaded with byte 8C. This selects sector 0, and sets the read condition. Thus, having previously been set to drive zero, track zero, it will now read sector zero. The accumulator will look for the incoming bytes in the memory location specified by the DE register pair (37EF). This is the memory-mapped location through which the actual data will flow.

The accumulator picks up the data from DE, stores it in the RAM memory location indicated by BC (4200); the next instruction increments register C so that location 4201 is ready. The program loops back, waits for another message from the FDC, picks up another byte, and stores it. When register C finally reaches zero, the pointer will again contain 4200, and the loop terminates. Then:

```

06C9      C3 00 42    JP      4200H

```

Here the disk system takes over completely. As you recall, starting at 4200, data from the disk has been stored. By jumping to that location, the program direction is wrested from ROM and given to the first 256 bytes of the disk system bootstrap program.

Here, then, is a quick review of the activity so far: Interrupts are disabled, cassette is turned off

and data are cleared from that output, video is restored to normal, and significant pointers for BASIC program operation are set up. A disk drive is searched for and if one is found, a group of procedures is initiated in order to transfer control of the TRS-80 to these disk instructions.

A series of control signals and acknowledgments is exchanged between the floppy disk controller and the CPU, a page (256 bytes) of data is poured into a RAM buffer area, and program control is given over to this new series of commands.

If a disk drive is not found, or if the break key is held down during power-up, control is transferred to address 0075. At this point it should be noted that the 'reset' button on the TRS-80 is a non-maskable interrupt, that is, the only interrupt which the DI (Disable Interrupt) command first executed by the TRS-80 cannot mask out. When pressed, the reset button goes directly to address 0066, following a much shorter series of instructions reminiscent of the power-up routine.

Because it is likely most important RAM pointers are still intact, this sequence does not reset them:

```

0066          ORG      0066H
0066      31 00 06    LD      SP,0600H
0067      3A EC 37    LD      A,(37ECH)
0068      3C          INC      A
0069      FE 02      CP      02
006A      02 00 00    JP      NC,0000
006B      C2 CC 06    JP      06CCH

```

This group of instructions sets up the stack pointer, checks for the presence of a disk drive, and jumps to the complete initialization routine (reboot) if it finds one. If none is present, it goes to the READY sequence beginning at address 06CC.

Now let us return to the initialization program flow we have been following, which is found at 0075:

```

0075          ORG      0075H
0075      11 B0 40    LD      DE,4080H
0076      21 F7 18    LD      HL,18F7H
0077      01 27 00    LD      BC,0027H
0078      E0 B0      LDIR

```

Using the LDIR instruction described earlier, a block of information located at 18F7 is transferred to RAM beginning at 4080. These bytes describe ports in use, error storage, INKEY\$ information, and so forth, as needed in the general operation of Level II BASIC (see Chapter 2).

A few specific addresses are delineated, and a large group of RAM bytes is then prepared. These jump to the familiar 'L3 ERROR' message because they are disk commands not available to Level II, yet patch points are prepared for them. The result of the following program statements is to fill addresses 4152 to 41A5 with the direction JP 012D.

008E	11 20 01	LD	0E,0120H
0091	06 1C	LD	8,1CH
0093	21 52 41	LD	HL,4152H
0096	36 C3	LD	(HL),0C3H
0098	23	INC	HL
0099	73	LD	(HL),E
009A	23	INC	HL
009B	72	LD	(HL),0
009C	23	INC	HL
009D	10 F7	OJNZ	0096H

Another group of ROM 'breakout' points follows; they all become returns to the main program flow. But notice something interesting about them – three bytes are set aside, but only one is filled with the return instruction (C9). This means that a jump command could be placed there. Let's first look at the series of instructions, then examine the possible benefits of changing them:

009F	06 15	LD	8,15H
00A1	36 C9	LD	(HL),0C9H
00A3	23	INC	HL
00A4	23	INC	HL
00A5	23	INC	HL
00A6	10 F9	OJNZ	00A1H

If we wanted to break into the BASIC operating system, this area of RAM is one place in which we could do it. Most of these are error codes of one kind or another. We could 'rescue' a program from displaying an error message, and halting, by patching in one of our own routines. If our routine were located at 5000, for example, the C9 instruction (followed by two unused bytes) could be replaced with a JP 5000 command, which needs all three byte positions: C3 00 50. Essentially, the authors of Level II BASIC provided many areas for expansion.

Now let's move on. BASIC programs begin at address 42E9. A pointer to that beginning is found as a zero at address 42E8. The next instruction sets that in place:

00A8	21 E8 42	LD	HL,42E8H
00A8	70	LD	(HL),8

The stack pointer is delineated, and a call is made to 1B8F, a subroutine to turn off or reset various devices, including the printer and cassette player. It is in part redundant, but a double-check is often worthwhile.

00AC	31 F8 41	LD	SP,41F8H
00AF	C0 8F 18	CALL	1B8FH
00B2	C0 C9 01	CALL	01C9H

The call to 01C9 results in the screen being cleared and the cursor being placed at position 0. Finally, 'MEMORY SIZE?' appears:

00B5	21 05 01	LD	HL,0105H
00B8	C0 A7 28	CALL	28A7H
00B8	C0 83 18	CALL	1883H

At address 0105 is a block of ASCII bytes which spell out MEMORY SIZE. The subroutine starting at 28A7 displays the string of data at the present location of the cursor, a byte at a time, until it finds a byte in the message whose value is 00. This terminates the display and advances the cursor. The call to 1BB3 is identical to the BASIC INPUT command, in that it displays the question mark and cursor, and halts for keyboard input.

If the keyboard input is the BREAK key, a carry is generated, and the program skips back to MEMORY SIZE and displays it again, waiting for keyboard input. The instruction RST 10 (ReStArt at 0010) follows, which is the quickest way of calling a routine to locate the first character of an input. If one is found, the result of an OR instruction will not be zero. Here are the instructions that perform those functions:

00BE	38 F5	JR	C,0085H
00C0	07	RST	10H
00C1	87	OR	A
00C2	20 12	JR	NZ,0006H

What if, on the other hand, there was no entry other than the ENTER key? You have no doubt noticed a slight pause in the action when you do not specifically set the memory size. Here's a look at that code:

00C4	21 4C 43	LD	HL,434CH
00C7	23	INC	HL
00C8	7C	LD	A,H
00C8	85	OR	L
00CA	28 18	JR	Z,00E7H
00CC	7E	LD	A,(HL)
00CD	47	LD	8,A
00CE	2F	CPL	
00CF	77	LD	(HL),A
00D0	8E	CP	(HL)
00D1	70	LD	(HL),B
00D2	28 F3	JR	Z,00C7H
00D4	18 11	JR	00E7H

For the moment we will start at the instruction LD A,(HL). HL contains the address of a byte of RAM memory, the contents of which are placed in the accumulator. From the accumulator, they are also saved in the B register. The accumulator is complemented, which inverts all the ones to zeros and all the zeros to ones. This complemented value is then placed in the

memory location still specified by HL. The accumulator is compared with what has been placed in HL.

What, you ask? But this value was just placed in memory, why compare it? Because – and this is a very elegant piece of writing – if it does not compare:

1. The memory location is bad and only the block of memory below it should be used to be safe.
2. Or, this is the end of memory.

If this is good memory, then, the test for zero passes, the contents saved in register B are returned to memory, and the program loops back, incrementing HL to the next potential memory location.

We did skip a few instructions back there. They become important only after the first loop is complete. These commands OR the contents of H and L; when the result is zero, we are at address 0000 – full 48k memory has been found, and the test is complete.

Here's what we would find, alternatively, if we entered some value (or other characters) in response to MEMORY SIZE?:

0006	C0 5A 1E	CALL	1E5AH
0009	B7	OR	A
000A	C2 97 19	JP	NZ,1997H
000D	EB	EX	DE,HL
000E	2B	OEC	HL
000F	3E BF	LO	A,BFH
00E1	46	LO	B,(HL)
00E2	77	LO	(HL),A
00E3	BE	CP	(HL)
00E4	70	LO	(HL),B
00E5	20 CE	JR	NZ,00B5H

The call to 1E5A checks for numeric input, and jumps to 1997 ('?SN ERROR'), if it is not numeric. If the input is properly numeric, then registers DE and HL are exchanged; this action puts DE (left off at the lowest usable memory location above pre-set RAM needed by BASIC) in HL, where it can be manipulated conveniently.

Memory size minus one is usable; memory size and above is protected. So HL is decremented before being tested, then it is tested (in a manner similar, but not identical, to that done earlier). If the memory test fails, it's back to displaying MEMORY SIZE? again.

We're not quite there yet, however, as the figure entered for memory size may be too small. BASIC needs a bit of room to work with, so DE is set to 4414, and the subtraction subroutine at RST 18 is called. If a carry is generated, we're shipped off to the '?OM ERROR' message found at 197A. Here's what it all looks like:

00E7	2B	OEC	HL
00E8	11 14 44	LO	DE,4414H
00EB	0F	RST	18H
00EC	DA 7A 19	JP	C,197AH

A little more work is left to do. Recall that a value for available string space is set aside, and it is 50 bytes. Here is how it is done:

00EF	11 CE FF	LO	DE,0FFCEH
00F2	22 B1 40	LD	[40B1H],HL
00F5	19	ADD	HL,DE
00F6	22 A0 40	LD	[40A0H],HL

Register pair DE is set up with FFCE, which, if you are not yet weary of manipulation of hex numbers, is the two's complement of 50 decimal. That is, when FFCE is added to 0000, the result is FFCE hex, or 50 decimal less than the original figure. Try it to see that it works. This bit of code saves the value for top of available memory in 40B1, adds register DE to it, and saves that result (memory size minus 50 bytes for string space) in address 40A0.

There follows:

00F9	CD 40 1B	CALL	1B40H
------	----------	------	-------

Here let me quote Roger Fuller, whose TRS-80 Supermap identifies this subroutine this way: Revelation 21:5 – "And behold . . . He shall make all things new."

This subroutine identifies and sets up all pointers necessary for the start of a BASIC program: Variables reset, previous program deleted, etc.

And now, the moment you've all been waiting for. Here it is:

00FC	21 11 01	LO	HL,0111H
00FF	C0 A7 2B	CALL	2BA7H
0102	C3 19 1A	JP	1A19H

The call to 2BA7, you may recall, displays a string of ASCII characters. The string displayed in this case is . . .

#### RADIO SHACK LEVEL II BASIC

The final instruction is a jump to 1A19, the address of the 'READY' display.

To summarize this last portion of the initialization routine: all BASIC pointers, disk error codes, and ROM return codes are set up, the screen is cleared, and the MEMORY SIZE prompt is displayed. A valid response to that question is accepted, and, if necessary, the entire bank of memory is tested. Error messages are generated as needed. Finally, the memory size and available room for strings is recorded, the READY prompt is displayed, and control of the TRS-80 is given to the user.

# NOTES

---

# 2

---

## Copacetic Comprehension

There will doubtless be a day when books like this will be unnecessary. Personal computers will probably develop into the appliance area, with programmers, hobbyists, hardware designers and language specialists present only in the distant background of the market. But between now and then we are all faced with being either frustrated users or solderer-programmers, tailoring machines according to our personal demands.

To do this, certain skills are inevitably required. Among these are an understanding of non-decimal number systems, digital logic devices, machine-level languages, and a smattering of diagnostic sense. There are some fine books that cover all these topics (see Appendix II), so this chapter will only deal with them as far as needed to put this book to work. Among them are:

- Binary, decimal and hexadecimal number systems, how they arose, how and why they can be used, and where understanding them is essential.
- Common digital logic devices that appear in the TRS-80 and these projects, and how and where to use them.
- Some of the basic elements of machine language, and a few personal considerations on where it is best applied, and when BASIC is a better choice.

- A look inside the TRS-80, with an eye to diagnosing where troubles might lie and where changes might be in order.
- The basics of creating a workable power supply for the projects in this book.

## Number Systems

Numbering is the single most overrated problem in computer programming. The answer (posed before the question) is this: numbers are merely *counting names*. That is, it makes no difference whether we think in tenths of a mile or eighths of an inch. Nor does it bother us that a day is made up of 24 hours, while an hour is 60 minutes. That a year is 365 days frightens us not, nor that months are a motley collection of sizes.

In parking lots, does it bother us that our vehicle may be parked in Row N as opposed to Row 14? There is no mystery when we mark off points with four scratches and a crosshatch. And does a dozen always conjure up 'twelve', or is a dozen something we have understood since youth?

Names are sizes are numbers; so it is with the number systems that we arbitrarily assign for the convenience of working with computers. When we are talking about electrical signals, it is clearest and easiest to think about ons and offs. Ons look pretty much like ones, and offs look like

zeros. It's a nice, clean concept, and one that illuminates the way we can refer to the machinery.

There's more convenience to naming a computer data condition 10110100 than to calling it an on off on on off on off off. Were data the only consideration, the binary one and zero method might have been satisfactory, without resorting to other means or stroking our memories.

Finding a location in a computer's memory is a much more difficult task. Although a memory location called . . .

11101000100110101

. . . might be easier to think about than . . .

onononoffonoffoffonoffonoffononoffonoffon

. . . it could use another step forward. In music, a long string of sixteenth notes like this –



Figure 2-1. Illustration of illegible musical notation.

– is broken up to make it legible, so it looks instead like this –



Figure 2-2. Illustration of legible musical notation.

Likewise, that long binary string can be broken up from 1101000100110101 into convenient groups . . .

## Converting Binary to Decimal

In the grade school years, students used to learn that a number like 5,163 contained a 3 in the ones place, a 6 in the tens place, a 1 in the hundreds place, and a 5 in the thousands place.

It was to remind them that 5,163 was really 3 plus 60 (6 x 10) plus 100 (1 x 10 x 10) plus 5,000 (5 x 10 x 10 x 10).

The way other number systems are written follows this same pattern for their own bases. In base eight the number 5,163 would have a 3 in the ones place, a 6 in the *eights* place, a 1 in the *sixty-fours* place, and a 5 in the *five-hundred-twelves* place. That means that 5,163 is really 3 plus 48 (6 x 8) plus 64 (1 x 8 x 8) plus 2,560 (5 x 8 x 8 x 8). But notice how that's decimal thinking! Really in base eight there could be no '8' . . . it would have to be called '10'! 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, and so on. So 5,163 in base eight is *still* 3 plus 60 plus 100 plus 5,000!

The binary system sneaks in the same way. A number like 1101 0001 0001 0011 turns into a 1 in the ones place, a 1 in the twos place, a 0 in the fours place, a 0 in the eights place, all the way up to a 1 in the thirty-two-thousand-seven-hundred-sixty-sevens place. In binary, the one on the far left is still a 1 in the quadrillions place, don't forget. But the message is how to convert all this to decimal. And here it is:

32768 16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1  
1 0 1 0 0 0 1 0 1 0 0 1 0 0 1 1

Just add the numbers: 1x1 + 1x2 + 0x4 + 0x8 + 1x16 . . . + 1 times 32,768 = 41,619. Voila. No matter how long the number is, and in whatever base:

1. Start at the left and produce a chart of the base number's powers, starting with 0 (X to the 0 power is always 1).
2. Lay the number to be converted underneath the base number chart.
3. Multiply each base number power by the digit in its place.
4. Sum the resulting numbers.

Does it work? Certainly. What is 163,341 in base 9? And in base 7? And in base 10?

Base 9 powers:	5	4	3	2	1	0
9 to that power:	59049	6561	729	81	9	1
Number to convert:	1	6	3	3	4	1
Multiplication:	1x59049	6x6561	3x729	3x81	4x9	1x1
Subtotals:	59049	+39366	+2187	+243	+36	+1
Converted result:	100882, base 10					

Base 7 powers:	5	4	3	2	1	0
7 to that power:	16807	2401	343	49	7	1
Number to convert:	1	6	3	3	4	1
Multiplication:	1x16807	6x2401	3x343	3x49	4x7	1x1
Subtotals:	16807	+14406	+1029	+147	+28	+1
Converted result:	32418, base 10					

Base 10 powers:	5	4	3	2	1	0
10 to that power:	100000	10000	1000	100	10	1
Number to convert:	1	6	3	3	4	1
Multiplication:	1x100000	6x10000	3x1000	3x100	4x10	1x1
Subtotals:	100000	+60000	+3000	+300	+40	+1
Converted result:	163341, base 10					

1101 0001 0011 0101

. . . although the legibility is improved, the human spark, the ability to look and recognize (that *aha!*) is not there. So the next step is to set about naming the sections. Since these on-off conditions can be written down as binary numbers, why not write them down in their decimal equivalents?

The question is rhetorical, of course, because not only can it be done, it is done. The only question is how to do it. Were a computer capable of swallowing all sixteen of those binary digits (bits) in one gulp, that question might be easily answered by calculating the conversion of 1101 0001 0011 0101 using a binary-to-decimal conversion table. The result, we find, is 53557.

But the computer, alas, cannot swallow all those bits in one bite . . . it can only swallow one byte full of bits (pardon). In other words, though a computer may need numbers sixteen bits long, only eight data lines exist to carry that data.

The component parts of the number 1101000100110101 are needed, eight bits at a time: 11010001 00110101.

There's the mathematical rub. 11010001 is 209 decimal, and 00110101 is 54 decimal. This seems hardly related to 53,557. Another solution is necessary, and it is a naming system as much as a

numbering system. It names each of the sixteen possible combinations of four binary digits:

## Reading the Pins

Finding your way through digital circuits is much easier than finding your way through an ordinary table radio. Industry standards have made the process simple. Consumer integrated circuits are packaged in small, rectangular, plastic or ceramic cases with anywhere from 8 to 40 external connections known as pins.

Earlier integrated circuits – and many of the audio types currently being produced – were packaged in small metal cans and looked like transistors, with many wires protruding from the bottom. The wires were arranged around a keying tab on the edge of the can, and numbered like so:

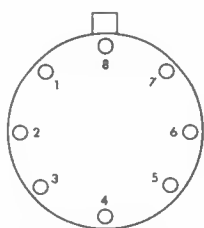


Figure 2-3. Can-type IC pin numbering.

As such circuits developed into more sophisticated and powerful devices, more pins were needed for input and output. A rectangular package was developed, but it was still numbered in a circle, starting (when looking down from the top) from the left of the notch, so:

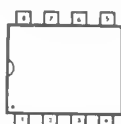


Figure 2-4. Dip-type IC pin number (8 pins).

All modern integrated circuits can be read from the top in this same way. 14- and 16-pin types start from the top left and read around:

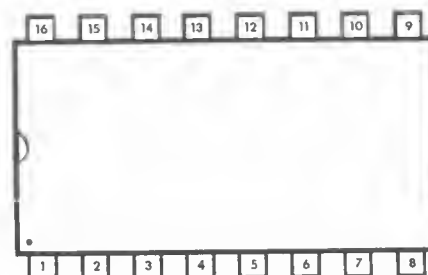
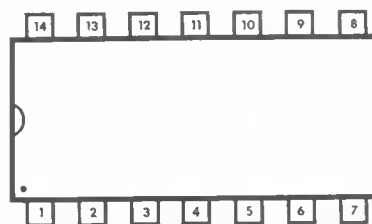


Figure 2-5. 14- and 16-pin Dip IC pin numbering.

You can read the pinouts of 18-, 20-, 24-, 28- and 40-pin circuits in the same manner. The highest numbered pin sits just opposite the lowest numbered pin. In the beginning this practice may seem confusing; it is. But after using the circuits – and counting their pins over and again – you will probably feel comfortable with the pin arrangement.

Just one thing: when you assemble TRS-80 add-ons, most of your work will be done from the bottom . . . which means reading backwards!

0000	is named	0	and is equal to decimal	0
0001	is named	1	and is equal to decimal	1
0010	is named	2	and is equal to decimal	2
0011	is named	3	and is equal to decimal	3
0100	is named	4	and is equal to decimal	4
0101	is named	5	and is equal to decimal	5
0110	is named	6	and is equal to decimal	6
0111	is named	7	and is equal to decimal	7
1000	is named	8	and is equal to decimal	8
1001	is named	9	and is equal to decimal	9
1010	is named	A	and is equal to decimal	10
1011	is named	B	and is equal to decimal	11
1100	is named	C	and is equal to decimal	12
1101	is named	D	and is equal to decimal	13
1110	is named	E	and is equal to decimal	14
1111	is named	F	and is equal to decimal	15

This may seem overdone; but A, B, C, D, E, and F are darn good names for binary values which exceed the number nine. If you don't have a name, make one up. For practical purposes, keep it within the symbols everyone has on the Royal typewriter.

Back to the number 1101000100110101. Crack it into those four legible pieces (1101 0001 0011 0101), and it can be named **D135**. To convert it to decimal, remember the old rule: the 5 is in the ones place, the 3 is this time in the sixteens place, the 1 is in the two-hundred-fifty-sixes place, and the D is in the four-thousand-ninety-sixes place. Thus, **D135** is 5 plus 3 x 16 plus 1 x 256 plus (see the chart) 13 x 4,096, or . . . 53,557!

So, that long binary number can actually be digested by the computer as a byte of **D1** and a byte of **35**. After a while, the number system comes easily. My personal recommendation: work in it. Convert to decimal only when you absolutely must. Think in hexadecimal and binary. They are the tools with which you can speak to the computer.

Throughout this book, numbers in hexadecimal are printed in **BOLD**.

## Digital Logic Devices

The binary number system and digital logic devices were developed together as a way of solving a practical dilemma: how to mass produce computers which could work quickly and accurately, and yet be inexpensive. As noted in Chapter 1, the problems of creating consistently accurate circuits, working with many different voltages levels, are formidable. Thus, simple yes-no, on-off logic was developed.

The intimidating term *Boolean algebra* is being used for the first, and last, time in this book – right in this sentence. You'll probably hear the phrase from time to time, but no matter – it's a professional's buzzword to keep the masses out. Forget it.

Back to digital logic devices. The essence of digital logic is to evaluate binary, on-off input; sometimes to determine a pattern of similarity or difference, sometimes to sense a change and sometimes to search for a signal. An appropriate result is produced as a result of the logical operation.

One of the logic building blocks is called a gate. A gate electronically evaluates its input to determine the pattern of similarity and difference of signals, and produces a specific output. A simple gate is shown below:

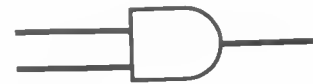


Figure 2-6. Simple AND gate.

Its job is to determine if the first AND second inputs are both at the one (high) level. Only under that condition will its output produce a high (one) signal. The table below shows how this AND gate works.

AND Gate		
If input #1 is	If input #2 is	The output result is
0	0	0
1	0	0
0	1	0
1	1	1

Table 2-1. AND gate action.

The table is called a *truth table*, and its purpose is to present every possible input and output condition for a given gate. Below is an OR gate. Stated in words, if either the first OR the second input is high, the output will be high. Examine the OR gate truth table; it really is quite logical.



Figure 2-7. Simple OR gate.

OR Gate		
Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Table 2-2. OR gate action.

Given a huge set of interconnected gates and their known inputs, the final output of the group can be determined by using truth tables like

these. Gates may have more inputs than two (some have sixteen), and may produce the opposite results from the two described above (NOT-AND and NOT-OR gates, known as NAND and NOR gates). Truth tables reveal how the integrated circuit's design engineer specified the pattern of binary logic inside the circuit.

In this way, given a desired output and a known number of input signals, it is possible to determine what set of input values will trigger the desired output.

There are a number of other types of digital circuits. Most are created out of gates like those described above, but their features are unique enough to think about them separately. Among these other digital logic circuits are buffers, flip-flops, counters, latches, multiplexers and shift registers.

A buffer can be thought of as a two-input gate with both inputs tied together, like this:

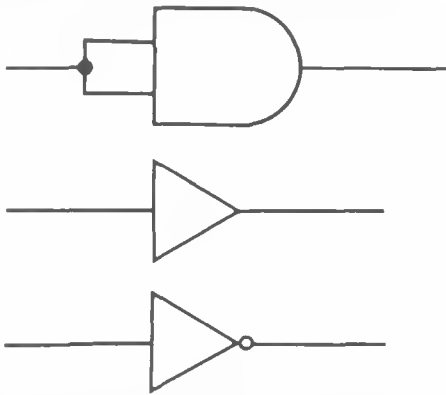


Figure 2-8. Buffer as (a) two-input gate and (b) buffer.

Its truth table is much simpler than that for two-input gates, because there are now only two input conditions. Either both inputs are high, or both inputs are low. Gates with 'true' outputs (AND, OR) will merely follow the input condition. When the inputs are high, the output is high; if the inputs go low, the output becomes low. Separate logic devices are manufactured that perform this 'follow-the-leader' function, and they are called *buffers*. They serve to isolate sections of a circuit, or rejuvenate a signal so it can feed many dozens of inputs in a large machine.

When a buffer reverses the condition of its input, (a high input is output low, and vice versa), the device is called an *inverter*. This kind of circuit can save the day in some cases, as when trying to locate a given binary number. Assume a circuit needs the binary number 1110 to turn on a pilot light. It is possible to choose four separate gates, each of which would provide an output

matching the desired number. These would be connected through more gates, and eventually the number could be discovered when the final signal was triggered properly. One way of detecting 1110 is shown below:

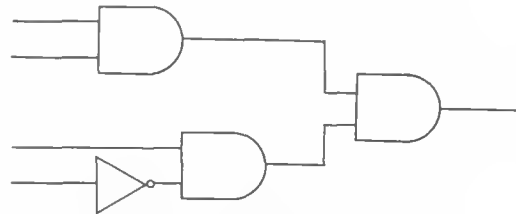


Figure 2-9. Bad decoding scheme for 1110.

But, although this circuit works, economy of cost and space and simple clarity dictate another solution. The last input could be inverted before it is evaluated, resulting in a pattern (1111) which could be quickly recognized by a multiple-input gate. The result is electronic simplicity and legibility; an improved decoding circuit is shown below. The ultimate result is the same.

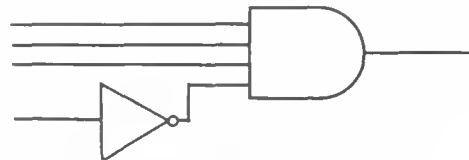


Figure 2-10. Good decoding scheme for 1110.

A flip-flop is a 'black box' which provides two outputs. When an input value is high (one), the first output will be high, and the second will be low. When the input value switches low (zero), the outputs will reverse. In other words, two opposite outputs for the price of one. But there is another significant use of the flip-flop.

Flip-flops also have an important input called a clock trigger, which is triggered only when its input returns to a given level. Only then will the outputs of the flip-flop reverse. That is, a given flip-flop clock may receive a 'zero' pulse. Its outputs will reverse. Then the zero pulse changes to a 'one' pulse. Nothing happens, but the trap is set to spring. When the one pulse changes back to a zero, the outputs reverse again. For every two changes at the clock, there will be but one change at the output. It takes four clock changes to produce two output changes.

Why is this useful? Because it is electronic, binary division. The truth table here shows how it works.

Binary Division with a Flip-Flop  
Output of First Flip-Flop Connected to Clock of Second  
Flip-Flops Change State Each Time Input Returns Low

Clock Input	Flip-Flop Output	Second Clock Input	Second Flip-Flop Output
0	0	0	0
1	0	0	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	1	1
1	1	1	1

(Input) (Input/2) (Input/4)

Table 2-3. Binary division with a flip-flop.

Digital logic devices known as *counters* are combinations of gates and flip-flops that allow certain patterns to be counted: Binary, Binary Coded Decimal (BCD, where the highest number is decimal 10), Gray code and others.

*Latches* are very much like flip-flops, except that the input is 'captured' at the output by a trigger signal called an enable, a select, or a gating pulse. The input may change continuously, but the output only reflects the input when the enable is activated. Latches are very useful when hundreds of thousands of signals are flying around on one set of lines, and the computer must select only certain groups of signals. The cassette output of data is a latch; only the 500-baud (bits per second) pulses of data reach the cassette output, even though many different signals reach its input.

*Multiplexers* are sometimes misunderstood, but mostly because of their formidable name. A traffic light is a multiplexer – it allows several streams of traffic to meet at one intersection, but only one stream to proceed. The multiplexer is the electronic equivalent, having several inputs.

Gating signals select which of the inputs may reach the output. In a computer, this allows several devices to share a circuit (like the video, which must be sent to the screen, but also sends and receives characters from the rest of the computer).

Finally, *shift registers* treat bits of data like a bucket brigade sends up water: it goes in one end, and at each electronic 'go!', the bucket is sent along one position. The dots which make up the video display are produced by circuits which shift them out to the screen one bit at a time, in synchronization with the monitor's sweeping electron beam.

## Into Machine Language

If we put faith in etymologists, then 'language' – which comes from the Latin meaning whole or part of a tongue – is an inappropriate word to put after 'machine'. Better to call it code, or blurms, or bingo chips even. Whatever it is, it becomes a tool for providing the user with all the power inside the TRS-80.

With a knowledge of binary and hexadecimal numbers, this language seems more fluid, and with a similar understanding of its electronic effects, it becomes the true 'lingua franca' of the computer.

In Chapter 1, I pointed out that BASIC is really just a disguised form of machine language. It is disguised because it presents itself in English-looking words, and has a large store of safety valves, error traps and messages – to prevent it from falling down an electronic rabbit hole.

For the moment I'll turn away from the metaphors, and present a practical simulation of that statement. The simulation can be familiarly written as follows:

### CLS

When you command CLS (clear screen), BASIC enters a machine language subroutine to clear the screen. It automatically returns to a BASIC command-level 'READY' condition. Now I happen to know that CLS is located at memory address 01C9 (457 decimal), and that (non-disk system) 'READY' is found at 06CC (1740 decimal).

As an experiment, type . . .

SYSTEM (ENTER)  
/1740 (ENTER)

. . . and you will be presented with the familiar 'READY', exactly as if you had pressed the Reset button. You've executed a machine-language GOTO, jumping directly to the 'READY' routine in ROM. As a second experiment, type . . .

SYSTEM (ENTER)

/1457 (ENTER)

Momentarily, the screen clears, but quickly crashes to MEMORY SIZE? Why did that happen? Notice that I said CLS was a *subroutine*. In other words, the routine must be called via some sort of GOSUB. The crash back to 'MEMORY SIZE?' occurred because there are no error messages in machine language unless you create them!

There are two ways to simulate the required GOSUB. The first is to POKE the starting

address of the clear-screen subroutine into the USR(0) command. The USR(0) command is identical to the SYSTEM command, except that it is the equivalent of GOSUB, where SYSTEM is the equivalent of GOTO.

So 01C9 (clear screen address) is the USR entry point; it must be split in two pieces (01 and C9), converted to decimal (1 and 201), and POKEd into USR addresses 16,527 and 16,526. So . . .

```
POKE 16527,1
POKE 16526,201
M=USR(0)
```

There. A screen clear without crashing to 'MEMORY SIZE?'. But there is another way. The machine language command for GOTO is called 'JUMP', and its hexadecimal code is C3. The machine language command for GOSUB is named 'CALL', and its code is CD. So here is the solution: CALL 01C9. JUMP 06CC.

Various hardware has its peculiarities. In the Z-80 microprocessor, addresses are always specified in reverse order. So CALL 01C3 is written CD (CALL) C9 (least significant byte of address) 01 (most significant byte of address). And JUMP 06CC is written C3 CC 06.

The whole process to clear the screen and jump to 'READY' is coded:

```
CD C9 01 C3 CC 06
```

Quite simple. CLS equals GOSUB 01C9, GOTO 06CC, which equals CALL 01C9, JUMP 06CC, which equals CD C9 01 C3 CC 06. The nice part of this little process is that it can be put anywhere in memory you like. Let's put it arbitrarily at 5000 to 5005 (20480 to 20485 decimal).

```
POKE 20480,205 : POKE 20481,201 :
POKE 20482,1
```

```
POKE 20483,195 : POKE 20484,204 :
POKE 20485,6
```

Convert the six hexadecimal bytes to decimal as shown earlier, and POKE them in place. Now you have the program completely under control. It is ensconced in memory, it calls the clear-screen routine, and jumps to 'READY'. Do you believe it? Just try it, entering the program at 5000. . .

```
SYSTEM (ENTER)
/20480 (ENTER)
```

There. A complete machine language program that functions from BASIC command level.

If you're new to this, take a break. The next step is to write an elementary screen-clearing routine, instead of calling one already in ROM.

I want to introduce a dilemma early on in the process of learning machine language. There is always discussion in computer programming about 'reinventing the wheel', and there is much truth to the suggestion that one should not do programming when the work has already been done. In the case of the screen-clearing program, the subroutine to do the work has already been created, and it is right there in Level II ROM waiting to be used.

Why, then, write another one? Why, in fact, even call the routine via machine language when the BASIC CLS command works so well? Indeed, with a BASIC as powerful as Level II, machine language should probably be reserved for doing things that cannot be done in BASIC at all. Among these things are upper/lower case drivers, programs to send characters to a serial printer, music-making and sound effects, telecommunications and so on. Furthermore, where machine language seems required, it often makes sense to call as many Level II ROM subroutines as possible.

Slick as this may be, it has two disadvantages: much of programming and customizing the TRS-80 requires an intense element of learning and understanding. Re-inventing the wheel is what everyone who learns must do, from the child who is forced through the memorization of 'times-tables', to the adult who has lost a job after 25 years and must learn new skills.

I can only present this from a highly personal point of view, as one who could not have learned machine language with any fluidity had I depended upon the software black boxes of others. I suggest that if you want a program to perform a certain action, try to write it in machine language. Try to make it do the same sort of error-trapping and other housekeeping that Level II's subroutines do. Take the Level II code apart and have a look. But don't deny yourself the opportunity to learn, rather than run a personal software assembly line. End of sermon. Back to clearing the screen.

The Z-80 microprocessor has internal holding latches called registers. Some are capable of holding eight bits, others sixteen. Some eight-bit registers may be paired up with others to create a single sixteen-bit register. These might be thought of as your only Z-80 machine code variables. In this screen-clearing routine, several registers will be used. The registers to be used in this experiment are: B, paired with C; H, paired with L; and A.

## What's In the Memory Map

The last of these is the *accumulator*, a sophisticated register capable of doing simple arithmetic. Indirectly, the 'condition code' register (called the 'flags') will also be needed, and will be described when it is used.

First, the program in its entirety:

```

21 00 3C      START—LD      HL,3C00H
01 00 04      LD      BC,0400H
36 20        LOOP—LD      (HL),20H
23          INC      HL
00          DEC      BC
..          LD      A,B
..          OR      C
C3 XX XX      JP      NZ,LOOP
```

Listing 2-1. Simple CLS demonstration.

In the first line, the H and L register pair is prepared with the values 3C and 00. As a pair, they are capable, then, of *pointing* to memory position 3C00. This is the memory location of the first position at the top left of the video display. The next line prepares the B and C registers with 04 and 00. Although they are pointing to memory location 0400, they are going to be used as a counter in this program, 400 is 1024 decimal, the number of places on the screen.

In the next line, the parentheses around HL mean 'the memory location defined by'. That is, the command is saying 'store the value 20 in the memory contents defined by the H and L register pair' — in this case, 3C00. So LD (HL),20 stores a blank space (20 is the ASCII value for such a space) at the first place on the screen. Whatever

was on the screen before is turned into a space.

Notice that this has the name 'LOOP' next to it. This 'label' is for the programmer's eyes, not for the program's use. In the next two lines, the HL pair is *incremented*. That is, 3C00 is incremented to 3C01. Correspondingly, the BC pair is *decremented*, from 0400 to 03FF. You may see the pattern emerging — BC is being used in a kind of machine language FOR-NEXT loop.

In BASIC, a FOR-NEXT loop is sort of self-testing. The programmer doesn't have to put in anything that checks the value of the loop variable; BASIC does it automatically. But in machine language, a test has to be made. The next two lines of the program make that test. The accumulator is prepared not with an absolute numerical value, but rather with whatever the B register reads at the moment. After one pass through, it will be 03.

Then the accumulator is asked to make a logical judgment. Recall that earlier in this Chapter, logical OR gates were discussed. If either of two inputs were high (one), the output would be high (one). In this case, there are *eight* inputs, one for each bit in the byte. So the accumulator has been loaded with 03, thus . . .

00000011

. . . and it is being asked for the logical OR with whatever register C is now set to. In the first loop, that is FF —

11111111

## What's in the Memory Map

The memory map of the TRS-80 has been designed for convenient, immediate use. It consists of seven major sections:

1. The BASIC language in read-only-memory (ROM).
2. An unassigned blank area for future expansion.
3. Special locations for cassette and disk.
4. A keyboard matrix, appearing in four locations.
5. A video memory.
6. A reserved block of RAM for BASIC use.
7. User-programmable RAM for programs, data, and variables.

The onboard jumper, X3, selects which of the various language possibilities is active in the TRS-80 (see section on electronic organization

below). Level I BASIC uses only 4,096 bytes from address 0000 to 0FFF, but Level II uses from 0000 to 2FFF. Because of hardware shortcuts, the cassette, disk and keyboard locations are incompletely decoded, appearing in 'phantom' areas beyond those strictly assigned to them. Using incompletely decoded locations in the cassette/disk area can result in unexpected results. But the phantom keyboards can be used interchangeably with the actual keyboard address.

The possible memory configurations of the TRS-80 Model I, together with reserved RAM areas, are shown below.

Figure 2-39. TRS-80 memory map in detail

Address	Function	RAM Pointer
0000	Beginning of all ROMs	
0000	Power-Up Position	
0FFF	End of Level I ROM	
2FFF	End of Level II ROM	
3000	Beginning of unassigned area	
3700	End of unassigned area	
370E	Disc Status Address	

370F	Disc Data Address	40A0	*1	String Space Pointer (2 bytes)
37E0	Interrupt Flip-Flop	40A2		Current Line in Use (2 bytes)
37E1	Disc Drive Select	40A4	*2	Start of BASIC Program (2 bytes)
37E4	Cassette Select	40A6		TAB Position Value
37E8	Line Printer Date	40A7	*3	Input Buffer Pointer (2 bytes)
37EC	Disc Controller Chip	40A9		Input #-1 Indicator
3800	Beginning of Keyboard	40AA		Seed Number for RND (3 bytes)
38FF	End of Keyboard	40A0		Reserved
3900	Phantom Keyboard	40AE		LET and DIM Scratchpad
3A00	Phantom Keyboard	40AF		Number Type Flag
3800	Phantom Keyboard	40B0		"Flag Byte for Encoder"
3C00	Beginning of Video	40B1	*4	Top of BASIC Memory (2 bytes)
3FFF	End of Video	40B3		String Scratchpad Pointer
4000	Beginning Reserved RAM	40B5		String Workspace (30 bytes)
4000	Restart #0 Patch (RST 8)	4003		Length of String in Use
4003	Restart #1 Patch (RST 10)	4004		Address of String in Use (2 bytes)
4006	Restart #2 Patch (RST 18)	4006		Next Available String Space (2 bytes)
4009	Restart #3 Patch (RST 20)	4008		State of Print Using (2 bytes)
400C	Restart #4 Patch (RST 28)	400A		Current DATA Line in Use (2 bytes)
400F	Restart #5 Patch (RST 30)	400C		OIM Scratchpad (2 bytes)
4012	Restart #6 Patch (RST 38)	400E		Print Using Scratchpad
4015	Keyboard Control Block	400F		SYSTEM Loading Entry Point (2 bytes)
401S	Keyboard Device Type (01)	40E1		AUTO On/Off Indicator
4016	Driver Entry Address (2 bytes)	40E2		Current Line in Use (2 bytes)
4018	Three Reserved Bytes	40E4		Size of AUTO Increment (2 bytes)
4018	Two Bytes Reading "KI"	40E6		Location of BASIC Command in Use (2 bytes)
4010	Video Control Block	40E8	*S	BASIC Stack Pointer (2 bytes)
4010	Video Device Type (07)	40EA		ERROR Line Number for RESUME (2 bytes)
401E	Driver Entry Address (2 bytes)	40EC		EDIT Line Number in Use (2 bytes)
4020	Location of Cursor (2 bytes)	40EE		Line Number before RESUME (2 bytes)
4022	Cursor Character	40F0		ON ERROR GOTO Line Number (2 bytes)
4023	Two Bytes Reading "00"	40F2		Reserved (3 bytes)
402S	Line Printer Control Block	40FS		Line Number Completed (see also 40E2)
4025	Printer Device Type (06)	40F7		CONTINUE Line Number (2 bytes)
4026	Driver Entry Address (2 bytes)	40F9	*6	Simple Variables Pointer
4026	Total Lines Per Page	40FB	*7	Arrays Pointer
4029	Current Line Being Printed	40FD	*8	Free Memory Space (FRE[A])
402A	Reserved Byte	40FF		Pointer to DATA in Memory
402B	Two Bytes Reading "PR"	4101		Variable Type Workspace (27 bytes)
4020	Level II Workspace	411B		TRON/TROFF Indicator
4020	Unassigned RAM	411C		Arithmetic Workspace (20 bytes)
403S	End Unassigned RAM	4130		Line/Print Using Buffer (33 bytes)
4036	Beginning Keystroke Storage	41S2		Disc Patch Points (see Chapter <?>)
403C	End Keystroke Storage	41A4		End Disc Patch Points
4030	Video Size / Cassette Latch	41A5		DOS Linking Patch Points
403E	Reserved for DOS Use (2 bytes)	41E7		End Linking Patch Points
4040	Storage Area for TIMES	<*3>		Keyboard/Edit Input Buffer
4047	End of TIMES Storage Area	42B8		Z-B0 Stack During Running Program
4048	Reserved for DOS Use	42EB		End Input Buffer
407F	End of DOS Reserved Area	<*2>		Beginning of BASIC Program
40B0	Storage Area for Division			End of BASIC Program
40B0	End of Division Storage Area	<*6>		Simple Variable Storage
40BE	USR Entry Point (2 bytes)			End of Variable Storage
4090	RND Storage Area (3 bytes)	<*7>		Array Storage Area
4093	INP Storage Area			End of Array Storage
4094	INPut Port Number (2 bytes)	<*8>		Free Memory Area
4096	OUTPut Storage Area			End of Free Memory Area
4097	OUTPut Port Number (2 bytes)	<*S>		BASIC Stack for NEXT, GOSUB, etc.
4099	INKEY\$ Storage Area			Top of BASIC Stack (works downward)
409A	ERROR Code Storage	<*1>		String Storage Area
409B	Line Printer Position			Top of String Storage (works downward)
409C	Output Device Indicator	<*4>		Top of BASIC Memory
4090	Video Line Length (32 or 64)	4FFF		End of 4K RAM
409E	Video TAB area	7FFF		End of 16K RAM
409F	Reserved	8FFF		End of 32K RAM
		FFFF		End of 48K RAM

— each bit in the accumulator is ORed with its corresponding bit in register C. If any pair of bits is 1, the accumulator's bit will be set to 1. When the ORing process is finished, the accumulator will contain the results, and the condition code register will reflect the meaning of those results.

Listing 2-2. Complete CLS demonstration.

```

00100 ; #####
00110 ; SIMPLE CLEAR-SCREEN DEMONSTRATION IN ASSEMBLER FORMAT
00120 ; #####
00130 ;
5000 00140 ORG 5000H ; BEGIN ROUTINE AT 20400
5000 21003C 00150 LD HL,3C00H ; VIDEO SCREEN, AT 15360
5003 01FF03 00160 LD BC,3FFH ; 1,024 SPACES ON SCREEN
5006 3620 00170 LOOP LD (HL),20H ; SPACE (DEC 32) IN PLACE
500B 23 00180 INC HL ; READY NEXT SCREEN LOC'N
5009 08 00190 DEC BC ; DROP SPACES LEFT BY ONE
500A 7B 00200 LD A,B ; GET CURRENT COUNT IN B
500B B1 00210 OR C ; OR WITH COUNT LEFT IN C
500C 20F8 00220 JR NZ,LOOP ; GO BACK IF NOT YET DONE
500E C9 00230 RET ; BACK TO MAIN ROUTINE
06CC 00240 END 06CCH ; READY AFTER TAPE LOAD
00000 TOTAL ERRORS

```

On the first pass, the accumulator will end up containing 11111111, and the condition code register's 'zero' flag will read 'not zero'. The last line of the program says 'jump if the result is not zero'. It jumps back to the part of the program marked LOOP, where it will store a 20 (space) in the new value pointed to by HL, increment HL again, decrement BC again, and go through the logical OR test once more.

If you carry the process through by hand, you will discover that only when B and C are *both* zero will the zero flag confirm a zero result. At that point, the program can shake loose from its loop.

Details of storage areas and their use can be found in *Microsoft BASIC Decoded, Supermap, Inside Level II*, and the *Level II BASIC Reference Manual*.

### Setting MEMORY SIZE?

Because machine language programmers have devised many unique ways of storing their programs, the purpose of responding to 'MEMORY SIZE?' has been the cause of some confusion. A look at the summary of the full memory map may help clarify the reasoning.

Reserved Memory	(not available)
BASIC Program Text	(fills upward)
Simple Variables	(fill upward)
Array Variables	(fill upward)
***** FREE MEMORY SPACE *****	
BASIC Stack	(fills downward)
String Storage	(fills downward)
MEMORY SIZE Value	(above reserved)

Table 2-4. Memory map summary.

This table points out two important facts: array variables grow *upward* into the free memory space. The BASIC stack (which stores GOSUB return addresses, levels of parentheses, FOR-NEXT information, etc.) grows *downward* into the free memory space.

Simple variables can also bump the array variables upward; string storage space is set ahead of time with the CLEAR statement. So you can see that the free memory area is impinged upon from both sides while a program is running. Although BASIC might have been designed to bump everything upward in memory (leaving the top area of memory unmolested), it would have resulted in considerably longer running time. This is because many changes in memory would have to be made when new variables, strings, parentheses, GOSUBs, etc., were discovered during a program's run.

If any machine language program is to be used, it certainly must be stored out of the way of this frantic activity. MEMORY SIZE therefore is used as a sacred boundary, above it is 'terra incognita' as far as the BASIC program is concerned. For example, if MEMORY SIZE in a 16K machine is set to 20480, the computer *acts precisely as if it were a 4K machine!*

To make maximum memory available for a running BASIC program, this boundary should be set only *just* low enough so that the machine language program will fit above it. Most program authors will write these programs to fit as high as possible in memory, and so you will normally see memory sizes (for a 16K machine) above 30000.

Why in a 16K machine would there be a 'memory size' of about 30000 and not 16000 or so? It's simply that the prompt 'MEMORY

SIZE?' is a bad question. The memory size value is really not a size at all, but the *address* of a memory location above which the BASIC program and its variables must not go.

Why then are there machine language programs which do not require memory size to be set? That is because clever programmers write machine language programs that . . .

- may be written for Level II BASIC, and thus can reside in one of the DOS reserved areas (see memory map).
- may automatically reset the memory size value before returning to BASIC.
- may be packed into strings where they are safely protected in a program text line (see Chapter 3).
- may be short enough to reside in part of the input buffer and change its pointer.

In all of these cases, something is sacrificed for the convenience of not setting 'MEMORY SIZE?'. In the first case, DOS-like expansion programs, such as Level III BASIC, will conflict. In the second case, programs which also require the memory size to be set may be damaged when the loading program automatically resets it. Thirdly, string-packed lines may not be edited without calamitous results. And finally, a reduced input buffer makes editing long lines impossible, as they will probably run into the BASIC program text.

So the 'MEMORY SIZE?' boundary is a useful feature of BASIC, serving to protect machine language programs from the expansionist tendencies of a running BASIC program.

### Comparing The Levels

Another source of confusion to a lot of users was the switch from Level I to Level II. How did this simple change of language alter the hardware? How did double-width characters, 500-baud tape loading, and key rollover suddenly appear? Why did the convenient abbreviations (P., N., M., F., etc.) suddenly go? Why were machine language programs happy with CLOAD in Level I, but needed SYSTEM with Level II?

The 4K BASIC in Level I is a compact, limited language with a few capabilities. Level II is three times as long, and much more powerful. Their authors, and hence their approaches, are different. The single hardware change in going from Level I to Level II is the installation of one ROM set in favor of another, and a minor change to allow 12K instead of 4K ROM to be accessed.

The 32-character mode hardware was already in place. The tape load speed and key rollover are all software controlled (see supplement to this Chapter). The abbreviations disappeared because Level II handled its keywords in a different manner from Level I, and such abbreviations would have increased execution time. Likewise, tape loading formats were a matter of design philosophy rather than any formal software requirements.

Level I has the advantage of being a simple, easily learned first language for computer beginners, and many TRS-80 owners learned by using that language. Level III is not a language distinct and apart from Level II, but rather an extension of the existing one. (In this sense it is much like Extended Color BASIC on the new TRS Color Computer, which does not supplant the original 8K BASIC, but merely adds another 8K to it).

What are the differences between the three levels? A command list for the three languages, with differences highlighted, follows:

Table 2-5. Comparison of Level I, II and III commands.

Command	Level I	Level II	Level III
@			
A. (ABS)	X		X
ABS	X	X	X
ASC		X	X
AT	X		
ATN		X	X
C. (CONT)	X		
COBL		X	X
CHR\$		X	X
CINT		X	X
CL. (CLOAD)	X		
CLEAR		X	X
CLOAD	X	*	X
CLOSE		X	X
CLS	X	X	X
CMD		X	X
CONT	X	X	X
COS		X	X
CS. (CSAVE)	X		
CSAVE	X	*	X
CSNG		X	X
CVO			X
CVI			X
CVS			X
D. (DATA)	X		
DATA	X	X	X
DEFDBL		X	X
DEFFN		X	X
DEFINT		X	X
DEFNG		X	X
DEFUSR		X	X
DEFSTR		X	X
DELETE		X	X
DIM		X	X
E. (ENO)	X		
EDIT		X	X
ELSE		X	X
ENO	X	X	X
ERL		X	X
ERR		X	X
ERROR		X	X
EXP		X	X
F. (FOR)	X		
FIELD		X	X
FIX		X	X
FOR	X	X	X
FRE		X	X
G. (GOTO)	X		
GET			X

GOS. (GOSUB)	X		X
GOSUB	X	X	X
GOTO	X	X	X
I. (INT)	X		
IF	X	X	X
INKEY\$		X	X
IN. (INPUT)	X		
INP		X	X
INPUT	X	X	X
INSTR			X
INT	X	X	X
KILL			X
L. (LIST)	X		
LEFT\$		X	X
LET	X	X	X
LSET			X
LEN		X	X
LINE			X
LIST	X	X	X
LOAD			X
LOC			X
LOF			X
LOG		X	X
LPRINT		X	X
M. (MEM)	X		
MEM	X	X	X
MERGE			X
MIO\$		X	X
MKO\$			X
MKI\$			X
MKS\$			X
N. (NEXT,NEW)	X		
NAME			X
NEW	X	X	X
NEXT	X	X	X
NOT		X	X
ON	X	X	X
OPEN			X
OUT		X	X
P. (PRINT,POINT)	X		
P.A. (PRINT AT)	X		
PEEK		X	X
POINT	X		
POKE		X	X
PDS		X	X
PRINT	X	X	X
PUT			X
R. (RESET, RND,RUN)	X		
RANDOM		X	X
REA. (READ)	X		
READ	X	X	X
REM		X	X
RESET	X	X	X
REST. (RESTORE)	X		
RESTORE	X	X	X
RESUME		X	X
RET. (RETURN)	X		
RETURN	X	X	X
RIGHT\$		X	X
RND	X	*	X
RUN	X	X	X
S. (SET,STEP)	X		
SAVE			X
SET	X	X	X
SGN		X	X
SIN		X	X
SQR		X	X
ST. (STOP)	X		
STEP	X	X	X
STOP	X	X	X
STRING\$		X	X
STR\$		X	X
T. (THEN,TAB)	X		
TAB	X	X	X
TAN		X	X
THEN	X	X	X
TIME\$			X
TROFF		X	X
TRON		X	X
USING		X	X
VAL		X	X
VARPTR		X	X
>	X	X	X
<	X	X	X
=	X	X	X
*	X	X	X
+	X	X	X
-	X	X	X
/	X	X	X
\$	X	*	X
{ }	X	X	X

\* indicates that Level I and Level II operations differ for this command.

## Hardware Reflects Software

You are probably familiar with the general operating characteristics of your TRS-80, including BASIC commands and how the machine responds to them. These responses are characteristics of how the software treats the hardware, and also of how aspects of the hardware act independently of the software.

The hardware inside the TRS computer can be broken into seven major sections:

### 1. CPU Hardware

- A. Central Processing Unit (Z-80), its clock, power-up, and reset circuitry.
- B. Decoding of CPU status signals into memory/peripheral access signals such as read, write, input, output.
- C. Buffering of address and data signals to and from the CPU.

### 2. Program RAM Control

- A. Refresh signals to maintain memory in dynamic RAMs.
- B. Address decoding able to distinguish 4K and 16K RAMs.
- C. Address multiplexing for dynamic RAM address lines.
- D. Read/Write signals to RAMs.

### 3. Video RAM Control

- A. Address decoding for video RAM.
- B. Read/Write signals to video RAM.
- C. Input to character generator, timing, blanking signals so characters do not run off the screen.
- D. Access management between display and CPU.
- E. Alphanumeric/graphic switching and graphics character circuitry.

### 4. Keyboard

- A. Address decoding for keyboard.
- B. Address/data buffering and read signals.

### 5. ROM Control

- A. Address decoding for 4K and 12K ROMs (Level I or Level II).
- B. Outboard decoding for three 4K Level II ROMs or two Level II ROMs.
- C. Read signals.

### 6. Output Controls

- A. Parallel-to-serial conversion from character generator.
- B. Horizontal and vertical video sync circuits, video output circuit.
- C. Cassette motor, cassette data, and 32 character video output control.
- D. Cassette audio output circuitry.

### 7. Power Supply

- A. Three regulated voltage outputs.
- B. Short-circuit protection.

Each of these sections plays a major role in the operation of the TRS-80, and few could be trimmed or eliminated without completely changing the character of the computer. The rest of this Chapter will be devoted to detailing those aspects of the TRS which are significant to customizing the hardware or software of the machine. For a more comprehensive examination, including timing diagrams and discussion of each signal line, turn to the *Technical Reference Handbook*.

## CPU Hardware

The master clock is produced by the oscillations of a 10.6445 MHz (million cycles per second, or Megahertz) crystal. A countdown circuit (Z56) divides this by 6, producing the running frequency of the Z-80, 1,774,083 clock cycles per second. This is generally called the TRS-80's 1.77-MHz clock.

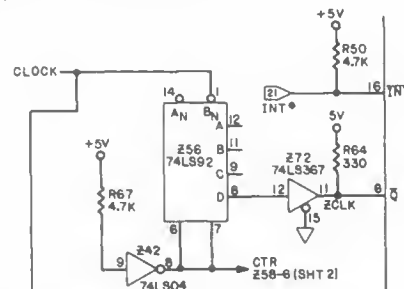


Figure 2-40. Clock circuit area of TRS-80.

It's interesting to note that there are several other clocks on the computer's board, some already wired in place, and others which can be created by interconnections. At pins 11 and 9 of Z56, 3.548 MHz is available, twice the normal TRS-80 speed. Pin 2 of Z43 clocks at 5.322 MHz, faster than the Z-80 can run, but when connected to pin 14 of Z56, a 2.661-MHz clock is available. This is 1.5 times the normal clock speed. Both the 2.661 MHz and 3.548 MHz clock rates will be used in Chapter 4.

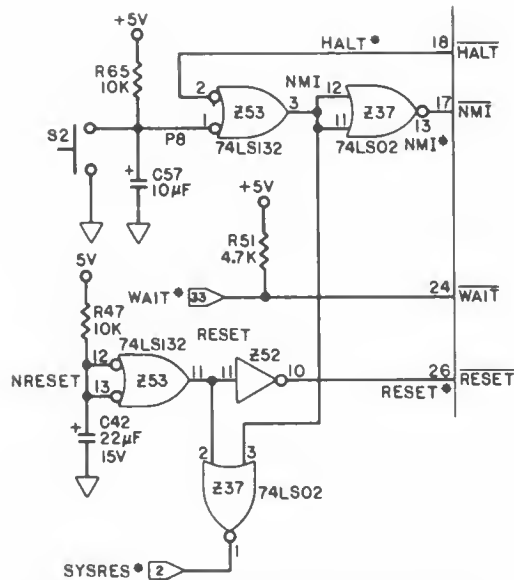


Figure 2-41. Power-up reset circuit area of TRS-80.

Upon power-up, the processor's RESET input, which sets the program counter running at 0000, is triggered by Z53 in combination with Z52. The capacitor C42 takes time to charge completely, so the RESET line is held down for a few milliseconds after the rest of the system comes up and is stable. This power-on reset is a convenience so the user doesn't have to press a special reset button just to get the system going. Other computers, such as the Ohio Scientific series, demand that inconvenient action.

The TRS-80 design creates Read (RD), Write (WR), Input (IN) and Output (OUT) signals from the Z-80's Memory Use Request (MREQ), Input/Output Use Request (IORQ), Write (WR), and Read (RD) signals. These are combined by Z23, in the correct order to do that. The four Z-80 signals *are not* wired to the edge card connector, so certain functions (such as mode 0 and mode 2 interrupts – see supplement to Chapter 5) cannot be used.

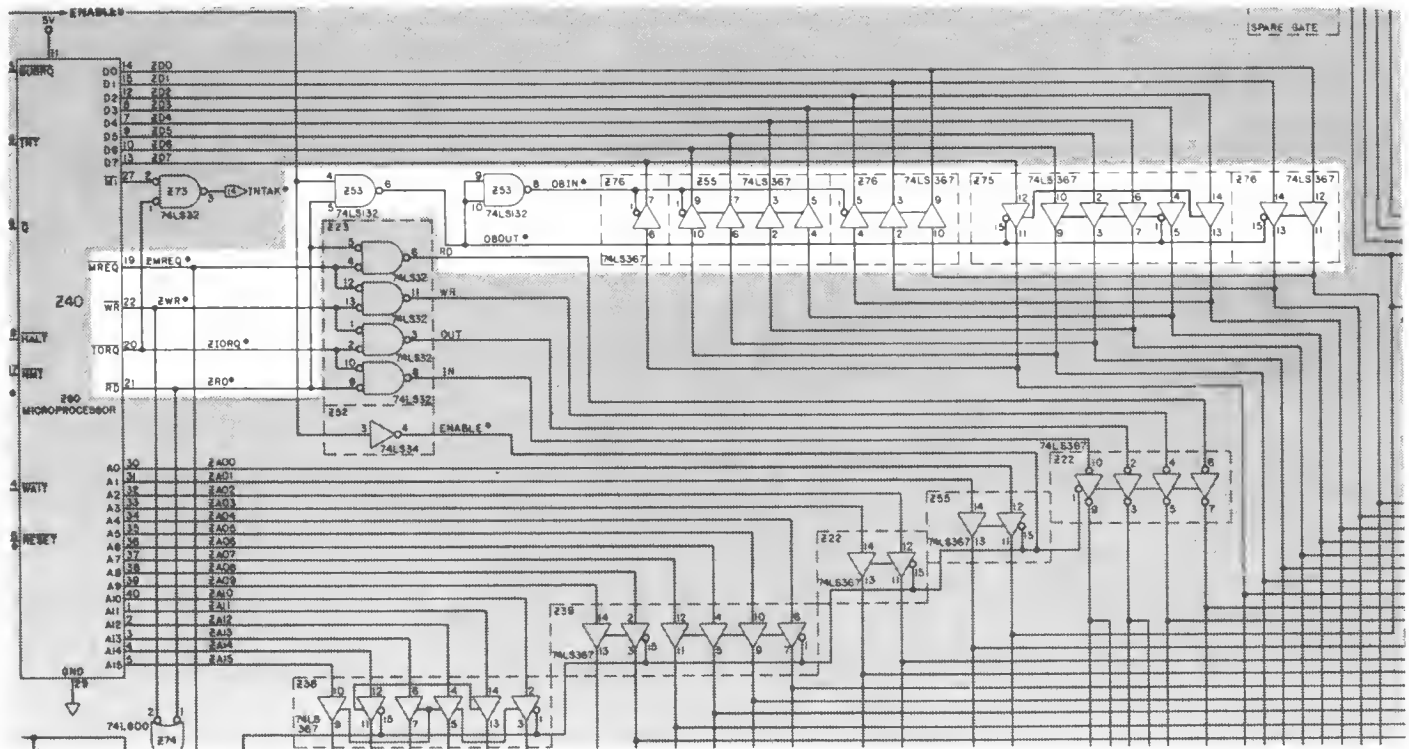


Figure 2-42. Read/write status, multiplexer of TRS-80.

## Program RAM Control

A TEST line is provided to 'float the bus' – in other words, the Z-80 becomes invisible, allowing another device to take over operation of the computer. Some outboard devices which speed up the TRS-80 use this feature, essentially taking control of the memory and peripheral devices by bringing the TEST line to ground. Z52 goes high in response, electronically disconnecting the Z-80's address and data bus from the circuitry. (Note that using the TEST line without memory-refresh backup circuitry on the outside of the computer will result in loss of memory contents).

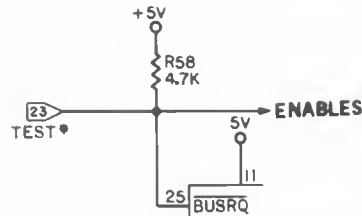


Figure 2-43. Test line circuit of TRS-80.

In normal operation, the Z-80 lines are active and buffered by Z38, Z39, Z22 and part of Z55 (address), Z75, Z76 and the remainder of Z55 (data). Except when the TEST line is used, the address buffers are always active. The data buffers are active under any circumstances, either in their READ or WRITE configuration:

Except for the memory refresh information, this completes the role of the Z-80 CPU circuitry in the TRS-80.

## Program RAM Control

The CPU is also used in the creation of the signals needed to refresh the dynamic memories. Since the TRS-80 uses dynamic RAMs, (see Chapter 5 for details on this), the normal refresh (RFSH) output of the Z-80 is less than useful, at least in the minds of the computer's design engineers. That RFSH signal, which is output when the computer is not using the memory, is ignored in the TRS-80.

Instead, the processor's MREQ line, when buffered, serves as a memory address row signal. The master clock is used in conjunction with the Z-80's RD and WR lines to produce a memory address column signal (column-address strobe, CAS) and a multiplex signal (MUX) to switch from row to column. This serves a very useful double purpose: not only does it refresh memory when the processor is not specifically using the memory in the program, but it serves as the address-select lines when the Z-80 processor is using memory. Refer to the Z-80 Technical Manual for details on the timing of these signals.

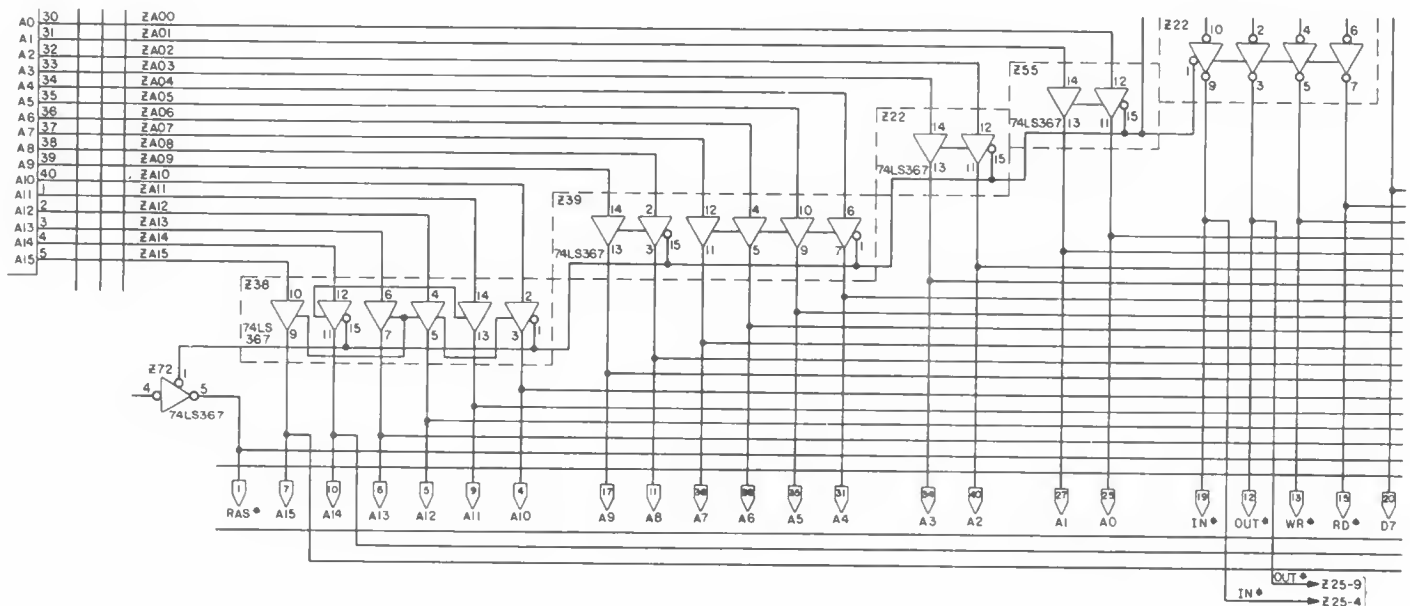


Figure 2-44. Address/data buffers of TRS-80.

Needless to say, should any of these signals fail or operate inconsistently, the memory will not retain its contents for very long *nor will a program even run which uses dynamic memory, because its address may not be selected.*

The next subject is memory management. This is basically the means by which the processor gets to the memory it wants to use. The heart of this sequence is found in Z35 and Z51, a pair of multiplexers which send the low bits of the memory address to the dynamic memories; flip from low bits to high bits according to the incoming multiplex (MUX) signal; and send the high bits to the memory. The memory, upon receiving these addresses together with the previously mentioned RAS and CAS signals, knows which address is being selected, and responds accordingly.

The DIP (Dual Inline Package) shunt Z71 plays an important role here. Specifically, it

routes the signals to the multiplexers in such a manner that the computer can distinguish between 4K and 16K RAMs. Stated simply, what is a memory-chip select line for 4K RAMs is a complete address line for 16K RAMs, and a partial address line for 8K RAMs (which the TRS-80 was designed to use also). Thus, the higher address lines must be prevented from appearing at the CE (chip enable) input of the 4K RAMs. If this were to happen, phantom memory would appear, and a running BASIC program (and the power-up memory test) might try to use those phantom bytes:

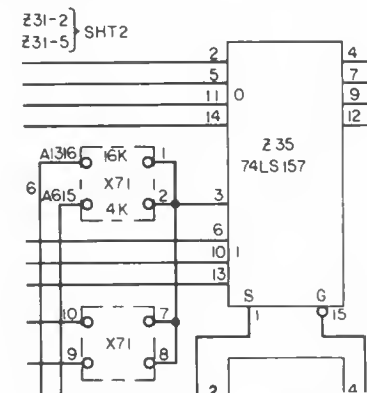


Figure 2-47. X71 and decoding scheme of TRS-80.

Only a WR signal is used to trigger these RAMs, as a high signal on their WR lines prepares them to read data. The CE (Chip Enable) determines if data should be placed on the data bus, where it is buffered by Z67 and Z68.

## Video RAM

Video memory is used in three ways: it is written to by the processor, read from by the processor and read from by the *video display circuitry*. In fact, it is constantly being read by the video display circuitry *except* when the processor demands attention.

The circuitry is complicated, and if you are interested in details, turn to the *Technical Reference Handbook*. In brief, memory is accessed by the processor using the Read (RD) and Write (WR) lines in conjunction with the decoded video area address-select line (VID). The display circuitry uses the video memory in a more complex manner: characters are output to Z27 and Z28, and these in turn are fed to Z29 (a character generator) and Z8 (a multiplexer). The characters, whether alphanumeric or graphic, are fed to Z10 (a shift register), where they are fed, a bit at a time, to the video output circuitry (beginning at the input of Z30).

Figure 2-45. Memory select/refresh of TRS-80.

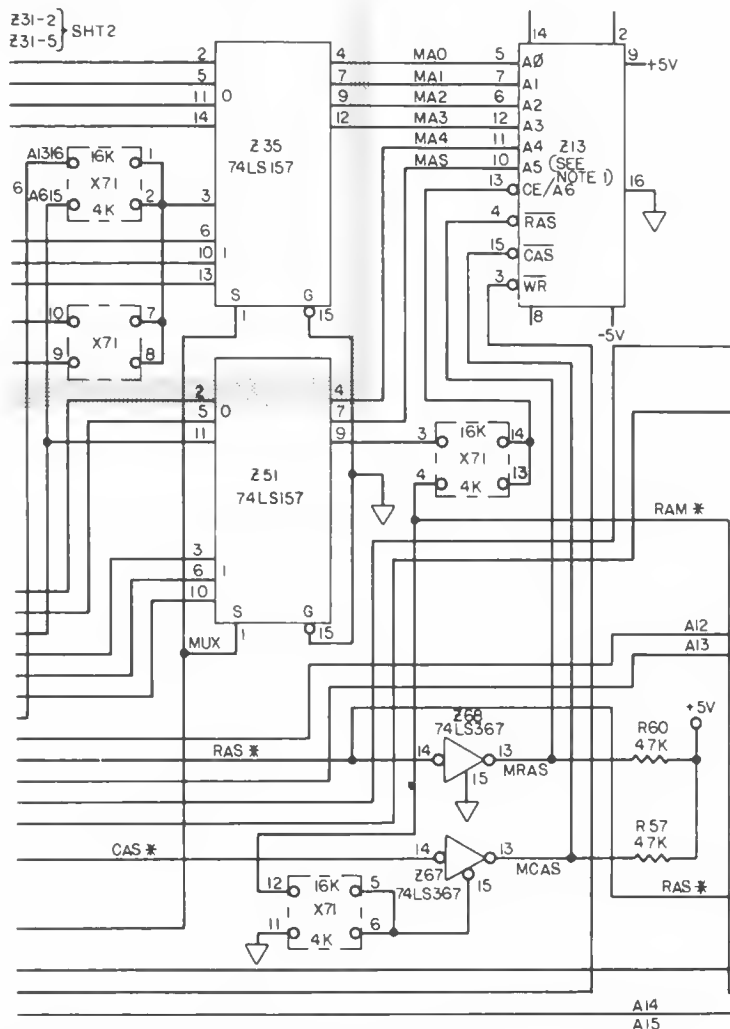


Figure 2-46. Memory multiplex, CAS, RAS & memory.

The character generation process is complicated by several factors. The dots that make up each letter must be fed to the video output circuitry only when the video monitor's electron beam is sweeping the visible part of the screen. The visible part of the screen does not include the upper, lower, left or right borders. The timing process must continue correctly even when the CPU is using the video memory.

Because each letter is made up of twelve vertical dots, and each line is made up of 64 characters with six vertical dots each, different parts of the characters must be output to the screen at different times.

Again, the *Technical Reference Handbook* covers this in detail, but a few decoded signals are important. The output of Z30, pin 10, is the final BLANK signal; no characters are output when this signal is active. Presence of characters or graphics in the border areas points to problems with this line.

The signal to shift video bits out to the video circuit is provided by Z26. Pin 8 controls alphanumeric bits, pin 6 controls graphics bits. Mangled screen characters may be traced to here, or to any of the seven chips that select characters: Z65, Z50, Z12, Z32, Z64, Z49, and Z31. This is one of the most unpleasant areas to attempt to diagnose.

The 32/64 character mode select (MODESEL) is provided by Z59, pin 9, and changes the speed of the video clock at Z43. Failures in either Z59 or Z43 will show up as a lockup in one mode or the other.

The presence of bit 7 determines if the computer is to output graphics or alphanumerics, and that signal (DLY BIT 7) is output in normal and inverted forms from Z27 pins 2 and 3. Failure in either mode can be examined here, or at the outputs of Z26, pins 6 and 8.

## Keyboard

The keyboard is very different from the video; it's just a simple key matrix. When the keyboard address area is read, the KYBD line from Z36 pin 11 triggers the keyboard integrated circuits (Z3 and Z4 on most keyboards) into action, outputting information to the data bus.

The data to be output is determined by the low eight bits of the address requested. The specific address requested is inverted, and a low signal is detected whenever a key in that matrix row is pressed. The inverting buffers to the data line provide the appropriate row information. For details on how the software interprets this switching matrix, see the supplement to this Chapter.

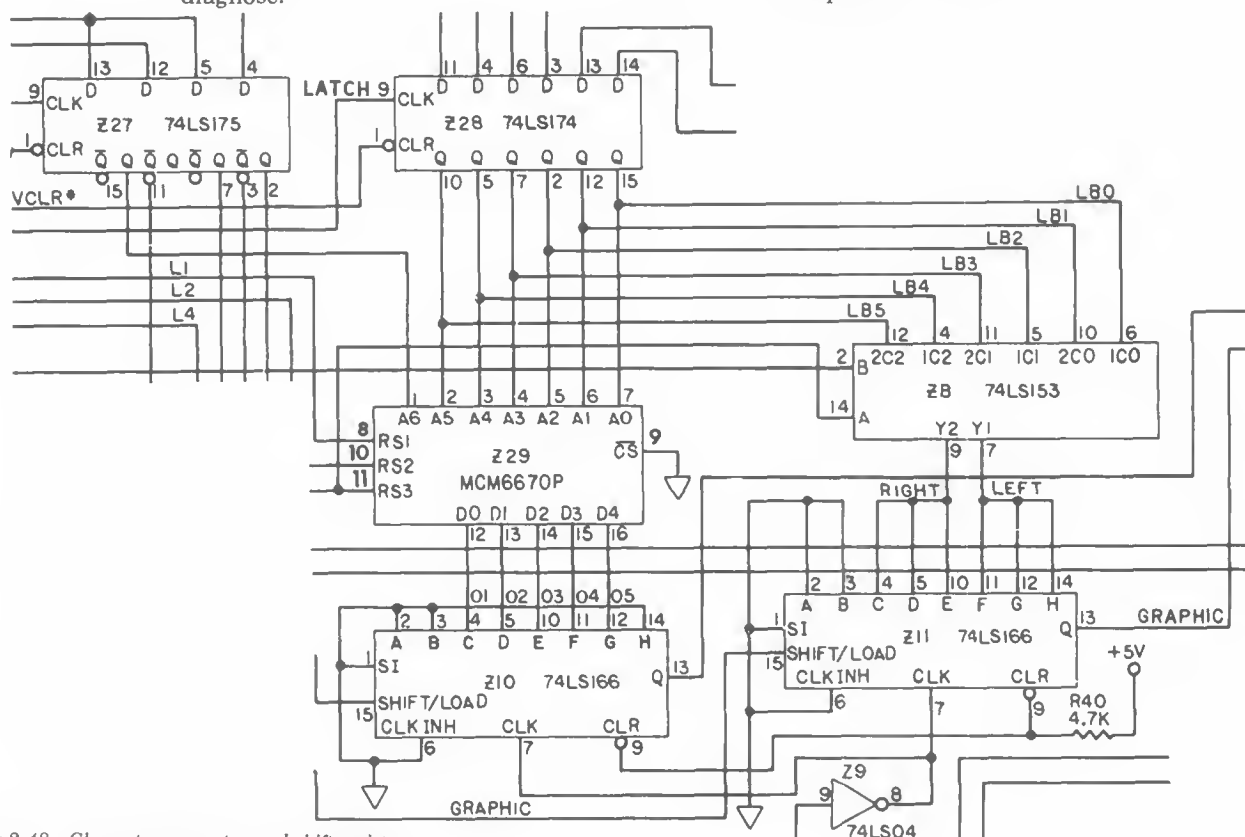


Figure 2-48. Character generator and shift register.

An interesting aspect to this is how the depression of *any* key may be detected. If address 387F is sought, all the address lines (except SHIFT) will become active, and the presence of any depressed key (except SHIFT) will appear on the address bus. Requesting the data at address 38FF will return the presence of any key *including* SHIFT. This is useful in creating a keyboard buffer, which is built from characters detected whenever the INTerrupt line is triggered.

In other words, the interrupt line triggers, and the program moves to the interrupt service routine. This routine reads address 387F, and the presence of any depressed key can be sensed. If one is pressed, it can be accepted and evaluated. Otherwise, the interrupt routine promptly returns to the program in progress.

This is also a valuable addition to INKEY\$ in some situations; see 'Peeking the Keyboard' in Chapter 3.

## ROM Control

Selecting the ROMs is the biggest sticking point in the Model I. This selection is accomplished by Z3, another DIP jumper shunt, in conjunction with Z21, a 74LS156 demultiplexer. Z21 joins various address lines to produce VID, KYBD, and MEM for the video, keyboard, and dynamic RAMs and to produce variants on the ROM line.

Two-chip Level I sets were selected by a combination of methods, all of which are detailed in the *Technical Reference Handbook*. Each ROM was 2K bytes in size. Three different versions of the board were publicly released, marked 'A', 'D' and 'G'. Each had a different hard wired method of decoding ROM. An occasional 'B' or 'F' board has been reported to me, but I have never seen one. Follow *Technical Reference Manual* descriptions carefully to make sense of these Level I lines.

Some of these ROMs were EPROMs with identical pinouts, so a 'ROM A' and 'ROM B' pair of lines were needed so these memories would not conflict.

Later Level I ROMs had the selecting circuitry masked right onto the chips, which removed that conflict. Finally, a single 4K ROM was introduced to eliminate these difficulties completely.

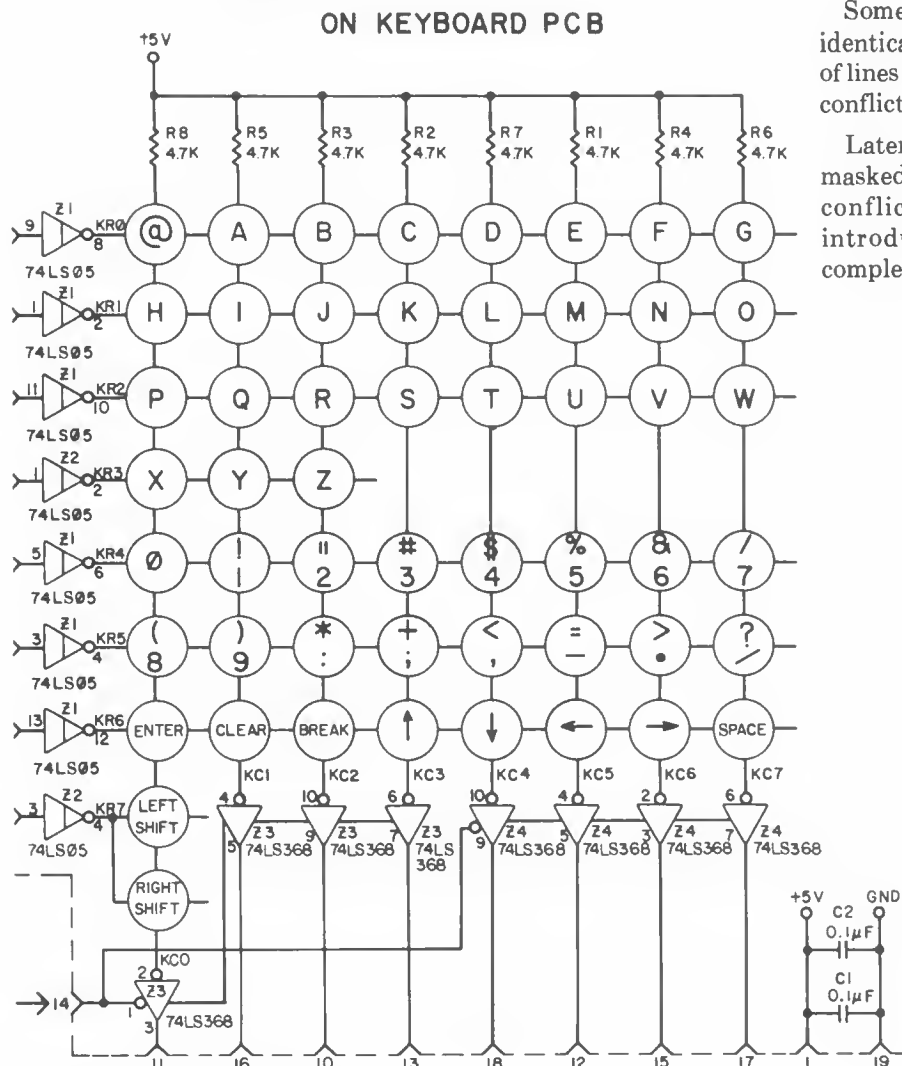


Figure 2-49. Keyboard matrix of TRS-80.

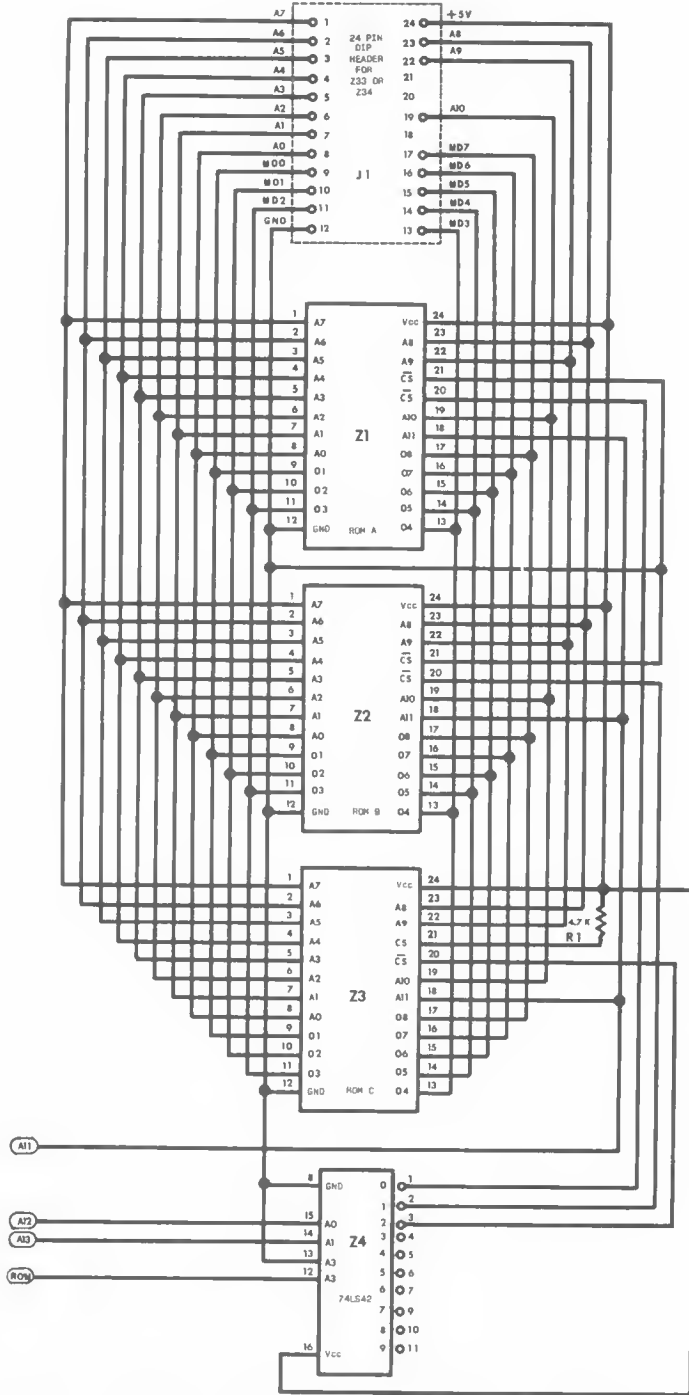


Figure 2-50. 74LS42 Level II demultiplexer of TRS-80.

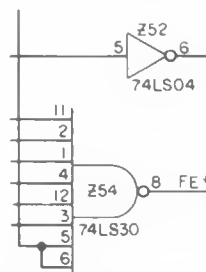


Figure 2-51. INSIG and OUTSIG area of TRS-80.

The Level II ROMs were for a long time quite consistent as a three-chip set, 4K bytes of memory in each one. A separate, piggybacked board was used. The decoded ROM line, in conjunction with address lines 11, 12 and 13, feed a 74LS42 wired as a demultiplexer. Each ROM selects a bit differently, and the combination of signals allows the memory areas from 0000 to 0FFF, 1000 to 1FFF, and 2000 to 2FFF to be selected.

In later Model I's, the three-chip set was reduced to two masked ROMs with different select characteristics. The piggyback board disappeared, and the two ROM sockets, once used for Level I, were now used for Level II. A modification to remove the two-chip ROM set, and make room for the addition of Level I, is presented elsewhere.

## Output Controls

Some of the output controls have already been pointed out, including the parallel-to-serial conversion of video characters and the 32 character video control. The latter is part of a latch which is activated by information sent to port FF. Z54, Z36 and Z52 together decode port FF, and this is brought together with the computer's IN and OUT signals. The result is a pair of lines marked INSIG and OUTSIG, which are used to activate the input/output circuitry.

A latch, Z59, holds the cassette motor on or off, and also locks the video into 32- or 64-character mode. It is also latched on and off to create the 500-baud pulse rate of data to the cassette player. This output signal is purely digital, as opposed to the input signal (discussed below).

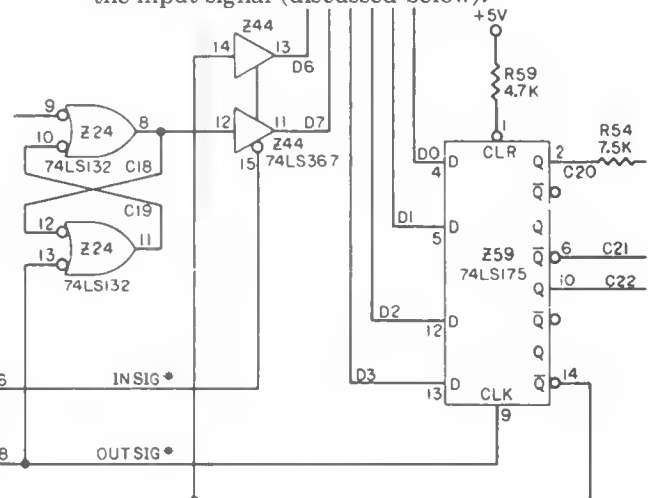


Figure 2-52. Output data latch area of TRS-80.

It would have been possible to have used Z54 to decode port FF directly, without using Z52 or Z36. Whether or not this was a design expansion consideration, you can use it as such since ports FF (255 decimal) and FE (254 decimal) are created by Z54 and the separate low-bit data line. In Chapter 4 port FE will be used for video and speed changes.

Cassette input is provided via a low-pass filter, through parts of Z4, where it is turned from low grade audio into a reasonable digital signal.

When INSIG is activated, whatever data is present at the cassette input is switched onto the data bus, and the CPU can read it. Then OUTSIG may reset the flip-flop created by Z24, when the program is ready to read the next piece of data from the cassette input. Note that the input can be any audio signal. The cassette port is not limited merely to taped data, but can be used to decode communications, shortwave, and ham transmissions, or test for the rise of voltage to a triggering level.

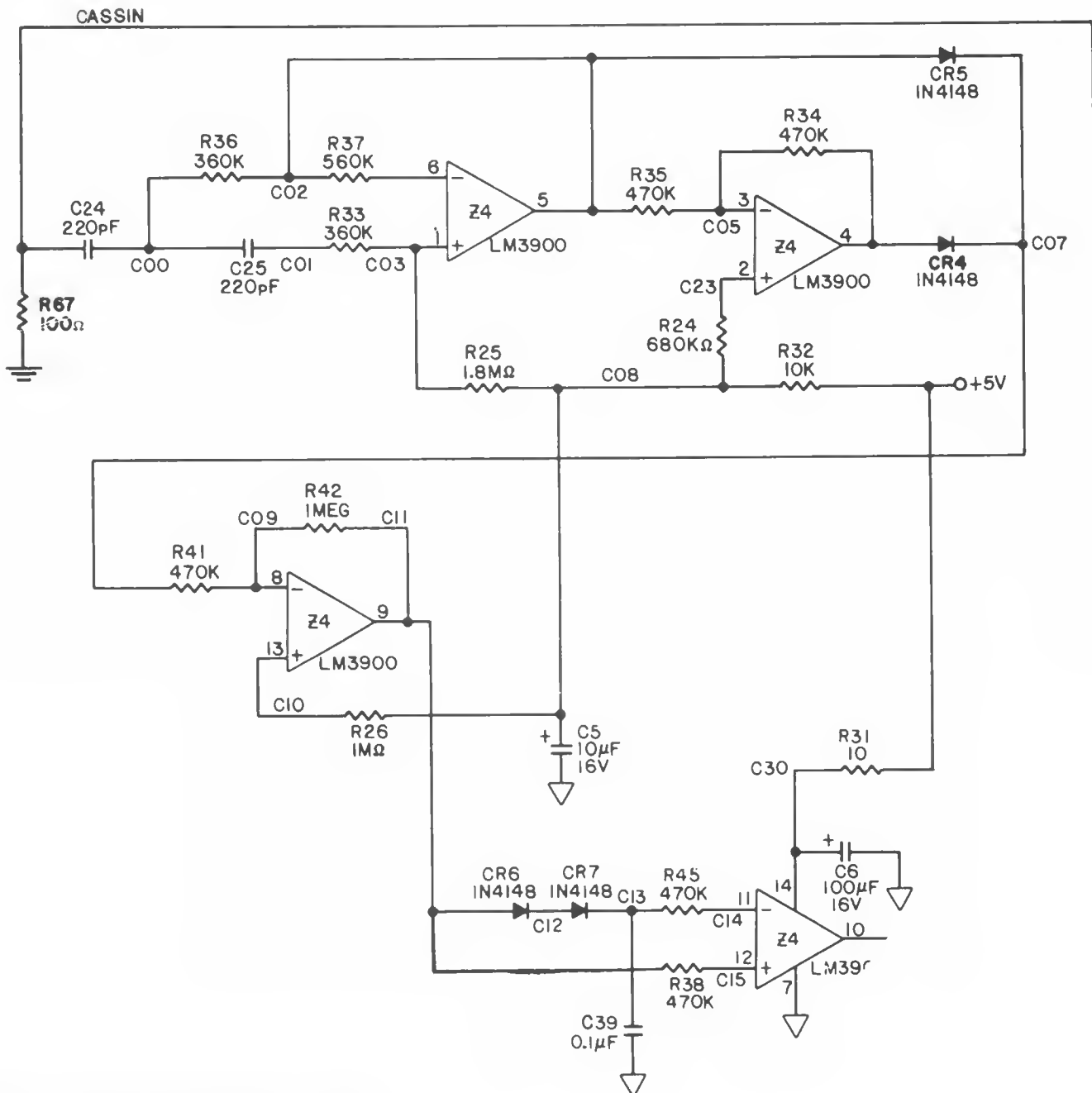


Figure 2-53. Cassette data shaping circuit of TRS-80.

The complex video divider chain provides HDRV and VDRV (horizontal and vertical drive) signals for television monitor synchronization. These signals are fed into a group of digital phase-shifting circuits which permit the signal to be adjusted on the video screen.

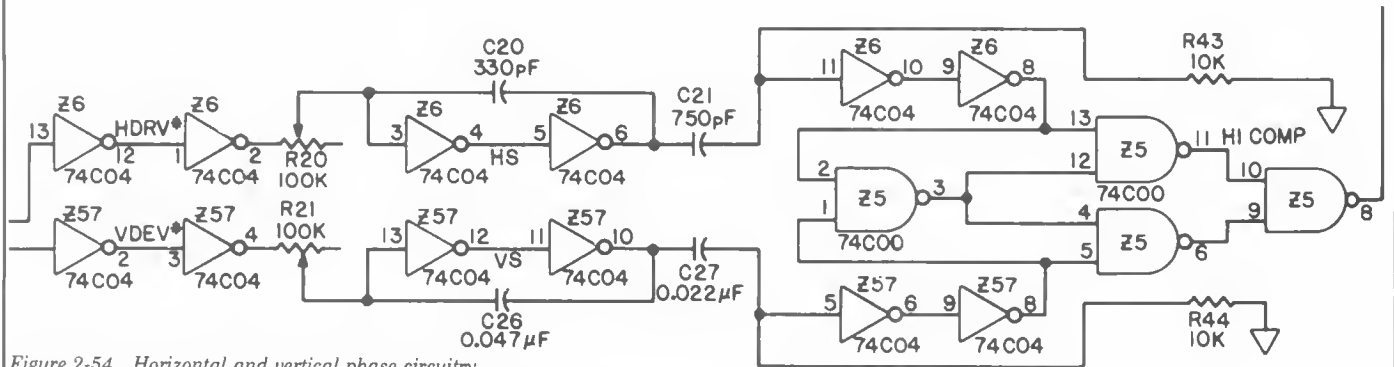


Figure 2-54. Horizontal and vertical phase circuitry.

The signals are mixed together at Z5 to provide a complete synchronization signal, and this sync signal is mixed with the video signal by Z41, Q1 and Q2. The result is a composite video signal which is capable of running a standard television monitor, or an RF modulator. The RF modulator signal can then drive an ordinary television.

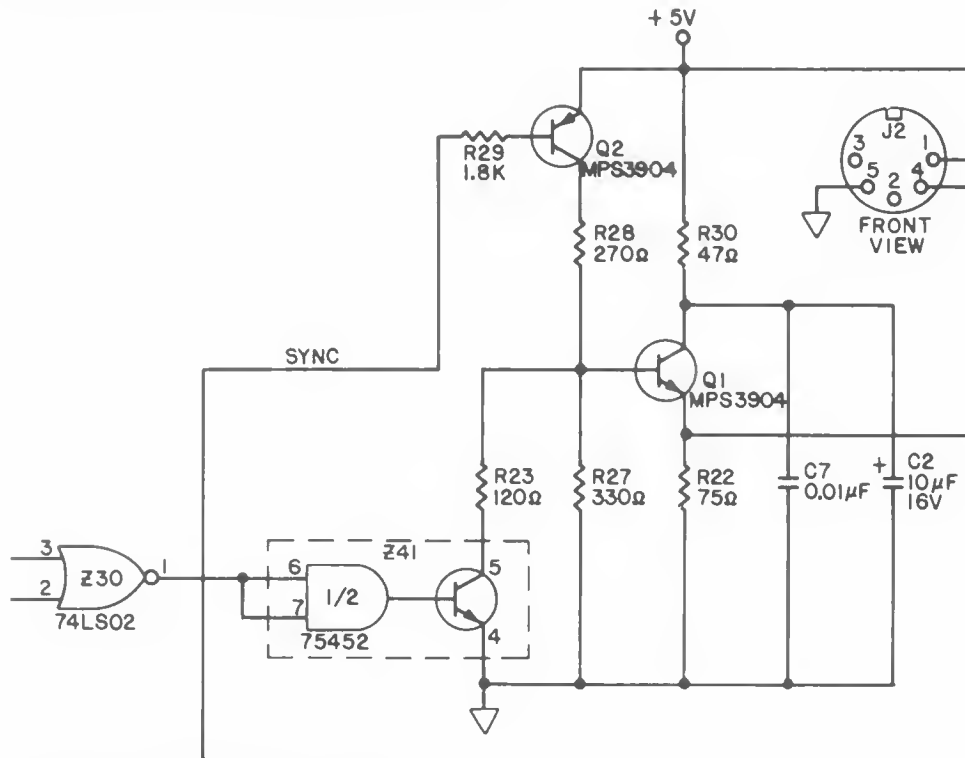


Figure 2-55. Video mixing circuit of TRS-80.

## Power Supply

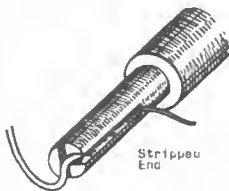
Refer to the *Technical Reference Handbook* for an excellent description of this circuitry.

## Wire-Wrapping Technique

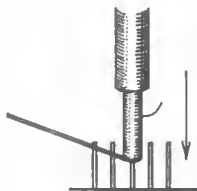
It's not without a bit of hesitation that I attacked many of the hardware projects presented in this book. Some are simple, but many, particularly those using memory circuits, need many connections. The wiring is not complicated, just tedious.

If you work carefully, all is likely to be well; but even a touch of haste will encourage confused connections. It is in these cases especially that wire-wrapping is the technique to use.

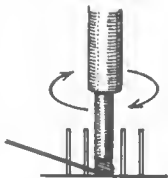
Wire-wrapping is not only easier than soldering, it is secure, simple, easier for correcting mistakes – and less costly. For wire-wrapping, you will need wire-wrap sockets, which are sold by most hobbyist supply houses including Radio Shack. Likewise, wire-wrap wire and a simple hand tool are used for the process. Here are the steps:



1. Insert stripped wire.



2. Slip over pin. Hold wire firmly, and slide fully down.



3. Spin wire – wrap tool. Wire rises along pin.



4. Finished connection has no bare wire protruding.

1. The wire, still connected to the spool, is inserted in the V-shaped stripping slot. Insert between one half and one inch of wire. Pull downward from the V, and the wire will slip out, leaving a piece of insulation in the stripper, where it can be shaken out.

2. Look carefully at the end of the wire-wrap tool. There is a small hole, meant to fit over the pins of a wire-wrap socket. Next to it is a half-circle, into which you must slide the stripped wire. The stripped portion will slide up a groove in the side of the tool, stopping where the insulation begins.

3. When the wire is in place, pull it sharply but gently upward, and slide the tool on the wire-wrap socket. Holding the wire firmly, spin the tool in your hand. The wire will wind up on the socket pin, freeing itself from the tool. Remove the tool. The wire-wrapping is complete for that end of the connection.

4. Cut the wire to a length that will comfortably reach its destination, and then some. Strip the end of it, and repeat the process above. The connection is complete. Don't forget to use different colors (white, yellow, red and blue are generally available). This will help you distinguish your connection patterns if changes become necessary.

## Peripheral Addressing

The bulk of the external devices attached to the TRS-80 do their own address decoding work. Some have become standardized by conventions of their use, and others have been used somewhat haphazardly by various manufacturers.

Table 2 - 5

Addresses reserved:

3000-37CF	Exatron Stringy-Floppy Personal Microcomputer Fastload Personal Microcomputer REX-80 Peripheral People Memory Siderer
37DE	Communications Status (Expansion)
37DF	Communications Data (Expansion)
37E0	Interrupt Flip-Flop (Expansion)
37E1	Disk Drive Select Latch (Expansion)
37E4	Cassette Drive Select (Expansion)
37EB	Line Printer I/O (Expansion) Percom Electric Crayon I/O Percom Speak-2-Me-2 Microcompatible Printer Buffer Floppy Disc Controller (Expansion)
37EC	Electronic Systems Serial I/O
37FB	

Table 2 - 7

Output Ports Reserved:

0	(Selectable) Alpha Product Interface Devices
1	(Selectable) Alpha Product Interface Devices
2	(Selectable) Alpha Product Interface Devices
7	(Selectable) Alpha Product Interface Devices
8	JPC Poor Man's Floppy System
55	Electronic Systems Serial I/O
208	Microperipheral Microconnection
209	Microperipheral Microconnection
232	Lynx Modem Radio Shack RS-232 Board
233	Lynx Modem Radio Shack RS-232 Board
234	Lynx Modem Radio Shack RS-232 Board
235	Lynx Modem Radio Shack RS-232 Board
254	Archbold High Speed Board Mumford Micro Speed Mod Board Most Internal User Modifications
255	Cassette Data I/O (Internal) 80-Grafix (Programme) Cassette Motor Switch (Internal) Video Character Size Latch (Internal) Simutak T-BEEP Addition
Addressable	Mullen Computer H-80 Interface Quent Systems PPI-80 I/O Port Orion Instruments In-Circuit Emulator Optimal Technology EPROM Programmer

Table 2 - 8

Other Peripheral Device Uses

Unaddressed (using bus control signals):

Cedret Software/Hardware Extension, The Patch  
Microprogramme Programmable Graphics, Grafix-80  
Microcompatible Company, The 225% Solution  
Alpha Product Stick-80 Joysticks  
SEL IEEE-488 to TRS-80 Interface  
Xitek STD Bus I/O Card System

Unaddressed (using cassette I/O signals):

Most available Light Pens, including  
Most available Cassette Data Digitizers, including  
Adu-Data, Data Dubber, E-Z Loader

Unconnected (using RFI interference pickup):

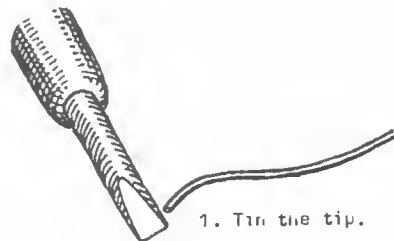
Micro-Mega CPU Monitor

## Soldering Technique

For projects from scratch, soldering should be considered the final process, the actions of a self-assured, confident hobbyist. For modifications, it is a necessity. In either case, and whether you are a micro-acrobat or distinctively clumsy like me, you can solder well. The requirements are patience and good solder.

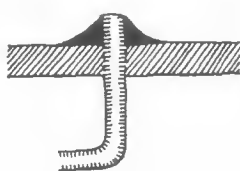
To start, make sure you are using an iron in the 25 to 40 watt range, never a soldering gun. The solder should be high quality, multicore solder. It is expensive, but will save many grief-stricken hours tracing 'cold solder joints', or removing globs of dull solder from between and under integrated circuits.

1. Clean the soldering iron tip, and heat the iron. Flow fresh solder on the tip to 'tin' the tip, which will help the solder flow from the tip of the iron to the part to be soldered. If the iron has been used, clean any encrusted material from the tip, and use coarse emery paper to shine the solder. If the tip gets deformed, bent, or very corroded, file it sharp with a fine file, and re-tin the tip.

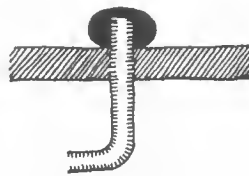


1. Tin the tip.

2. Keep an old sponge handy, slightly damp. Run the tip of the iron quickly over it as you solder to remove the excess flux. Always use a soldering iron holder (usually provided with an iron); if you don't, you'll wish you had the first time you burn a large hole in your imitation walnut, vinyl-topped desk.

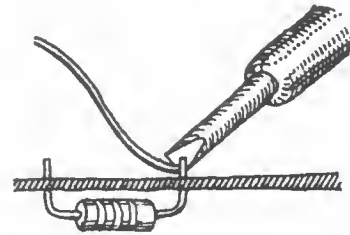


4. Finished solder connection.



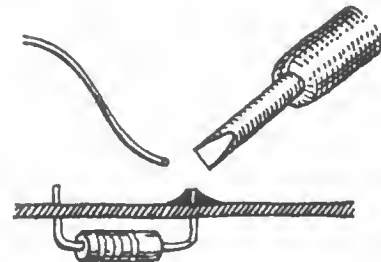
5. Bad solder connection - no contact with board (sideview).

3. In the olden days, the rule was 'heat the parts, not the solder'. Forget it. Make sure the iron is no hotter than 40 watts (and remember never to use a soldering gun) and that the parts you are about to solder are very clean. Place the iron against the part, making as much contact with it as possible along the angled tip of the iron. Place the end of the solder at the juncture of the iron and the part, and flow just enough solder to make a clean, shiny, flowing connection.



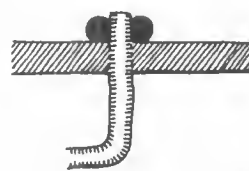
2. Bring solder, parts and iron into contact.

4. Remove the iron immediately and let the part cool. If a wire is being soldered, hold it still until the solder becomes cloudy and cool, or else an incomplete connection may result.

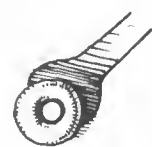


3. Lift iron and solder simultaneously.

5. If solder bridges develop between connections that are very close together, don't try to suck up the solder with the iron; you can only overheat the parts that way, and end up with blobs of solder and flux. Instead, use solder wick or solder-up to remove the excess solder, and start again. Let the parts cool before soldering again (a half minute should be enough).



6. Bad solder connection - no contact with pin (6A sideview) (6B topview).



## On the Keyboard Scan

Arrows? Control codes? Autorepeat? Whatever it is you would like, that has to do with the keyboard, you can do with the TRS-80. The designers of the machine chose not to use an ASCII (American Standard Code for Information Interchange) keyboard . . . one that outputs a code for each key pressed; instead, the keyboard is a matrix of switches. Because of this decision, the TRS-80 keyboard can be extremely versatile with a minimal body of software.

First let's take a look at the keyboard matrix itself. If you have been programming in machine language, or even relatively sophisticated BASIC, this map will be familiar:

Addresses	Keys															
3801	@	A	B	C	D	E	F	G								
3802	H	I	J	K	L	M	N	O								
3804	P	Q	R	S	T	U	V	W								
3808	X	Y	Z					RESERVED								
3810	0	1	2	3	4	5	6	7								
3820	8	9	:	*	;	+	<	=	.	>	/	?				
3840	ENT	CLR	BRK	UPAR	DNAR	LFAR	RTAR	SPC								
3880	SHIFT							RESERVED								
Data:	01	02	04	08	10	20	40	80								

At first, the arrangement of the address lines and data information may seem unappealing. What is the use of having address and data information that doubles at each change? Why not just use 3801, 3802, 3803 and so on? The reasons will eventually become apparent – in the ease and speed of the keyboard scan (and in the low cost of the hardware), but consider this binary for a moment . . .

```
00000001 00000010 00000100 00001000
00010000 00100000 01000000 10000000
```

. . . and there you have it. The bit is bumped along for each keyboard row and column, so that the presence of a single active bit instantly identifies any of the (potential) 64 keys.

The entrance to the keyboard scan is made with a jump from address 4016 in the ROM's RAM switchboard. Normally at power up, the jump address 03E3 is put in place, and the keyboard scan is entered at that point. Registers BC, DE, HL, and A are used in the scanning process:

```
03E3 21 36 40 LD HL,4036
03E6 01 01 38 LD 8C,3801
```

The HL register pair points to the first RAM location at which the keystrokes will be stored. BC is set to look at the first row of the keyboard, whose memory map is 3801 to 3880 as noted earlier. Register D is set to zero, and it will become a 'row counter'; the process begins at

address 03EB:

```
03EB 0A LD A,(8C)
03EC 5F LD E,A
```

The accumulator reads the data at BC (recall at the outset it is pointing to the first keyboard row, 3801). The information it finds is stored in the memory location pointed to by HL (4036).

Okay so far. Now comes some of the interesting stuff that distinguishes this scan as an excellent piece of writing:

```
03ED AE XOR (HL)
03EE 73 LD (HL),E
03EF A3 AND E
03F0 20 08 JR NZ,03FA
```

This short segment is responsible for the 'rollover' capabilities of Level II. The contents of the accumulator (the keystroke, if found) is XORed with the previous contents of 4036. Recalling how the XOR function works, we discover that if the key pressed was the same as the previous one at this row, the accumulator will be 'toggled' to zero. In any case, the current keystroke, whatever it is, is now saved in 4036 (so that the next time 'round, it knows if a key is still pressed).

If the key was the same, AND E will be the result of A-toggled-to-zero AND the found keystroke . . . or zero. If there was no key pressed, the result will be A XORed with HL (which is essentially irrelevant) AND E-which-is-zero . . . or zero. The test at 03F0 is for not zero. Under these conditions, it fails, so the program continues:

```
03F2 14 INC D
03F3 2C INC L
03F4 C8 01 RLC C
03F6 F2 E8 03 JP P,03EB
03F9 C9 RET
```

The 'row counter' (D register) is incremented, and the low-order byte of HL is incremented (to storage address 4037), and the low-order byte of BC is rotated. Recalling the keyboard matrix, we can see that this command to rotate moves us from 01 to 02, from 02 to 04 from 04 to 08, from 08 to 10 and so on. That keeps track of the row that the scan is looking at, and as long as the result of the rotate is positive (bit 7 low), the loop will travel back to 03EB, where the next row will undergo the same testing as each previous one.

When RLC C shifts the row pointer to 3880, then bit 7 will be high (10000000); this is 'negative' in Z-80 architecture, and the loop falls through. Why does it fall through before checking the contents of address 3880? Because the only thing in this row is the shift key; it does not offer a decipherable code by itself, but merely modifies the information found when some other key is depressed. This explains why, among

other peculiarities, INKEY\$ does not acknowledge SHIFT alone.

When the loop falls through, the program encounters a RETurn from subroutine, which directs it immediately back to the rest of BASIC. The routine is remarkable, looping through just over 100 bytes when the keyboard is clear. Although not as time-efficient as obtaining input from a memory-mapped ASCII keyboard, it is quite speedy, and offers considerably better 'rollover' than many encoded keyboards.

When a key is pressed, the program jumps to 03FA, and is able to provide upper/lower case ASCII codes, special functions, and, incredibly enough, all of the 'missing' ASCII control codes (form feed, ring bell, etc.). Let us now follow it through:

```

03FA 5F      LD    E,A
03FB 7A      LD    A,0
03FC 07      RLCA
03FD 07      RLCA
03FE 07      RLCA
03FF 57      LD    0,A

```

The position of the keystroke found has been stored in register E – recall that this is the 'column' of the keystroke. The row itself is not yet accessible, but the row counter (register D) is crucial to determining it. After E is saved, the accumulator is loaded with the value in this row counter, and rotated to the left three times. For those shaky in their binary arithmetic, this is the effect: if a decimal number is 045, a left rotation makes it 450. This is multiplication by ten. If a binary number is 010 (decimal 2), a left rotation gives 100 (decimal 4) . . . in other words, multiplication by two. Therefore, three left rotates gives us 2 x 2 x 2, or multiplication by eight. That result is saved back in register D.

The purpose of this clever play will soon become clear:

```

0400 0E 01      LD    C,01
0402 79      LD    A,C
0403 A3      AND    E
0404 20 05      JR    NZ,040B

```

Here the C register is set to 1, sucked up by the accumulator, and ANDed with E (remember E still contains that keystroke column byte). If the result is not zero (that is, if E equals 1), then the loop falls through and the program moves on. But have a look at what follows:

```

0406 14      INC    0
0407 CB 01      RLC    C
0409 1B F7      JR    C,0402

```

What is this about? Well, the D register, which contains 8 times the row value, is being incremented each time C is being rotated . . . making the lower three bits of D serve now as a column counter! Whoa, you say, back up there.

Okay, here it is: the original value in D could have been 0 through 6, depending on the row in use. When shifted three times, the possible values become 00, 08, 10, 18, 20, 28 and 30. Each of these possible values, when incremented through all seven possible columns, might contain 00 to 07, 08 to 0F, 10 to 17, etc., up to 37. This gives us a complete, distinct value to represent each key.

Now a fairly crude process of hunt-'n-peck begins. The status of the SHIFT key is checked, and set aside in register B. The demultiplexed keystroke value in register D is snapped back into the accumulator, and the comparisons take off:

```

040B 3A 80 38      LD    A,(3880)
040E 47      LD    B,A
040F 7A      LD    A,0

```

The character search can be followed through several branches; we will start with the most straightforward, and progress through some of the unique (and little publicized) aspects of the TRS-80 keyboard output.

The program adds 40 to the character value (address 0410), and checks if the result is greater than or equal to 60 (0412).

```

.....
. CHAR + 40 is      @ A B C D E F G .
. less than 60      H I J K L M N O .
. (40 + 00 to 1F)   P Q R S T U V W .
.                  X Y Z .
.....
. CHAR + 40 is      0 1 2 3 4 5 6 7 .
. 60 or greater     B B :* ;+ ,< = > /? .
. (40 + 20 to 37)   ENT CLR BRK UPA DNA LFA RTA SPC .
.....

```

If the compare finds a value less than 60, the routine rotates the SHIFT key value – which had been saved in the B register (0416). If SHIFT is released, the value in B is zero, and hence the rotate resets the carry flag (0418). The program moves directly to the terminal steps at 044B (to be discussed later). At this point, the character contained in A would be in the range 00+40 to 1F+40, the ASCII values for upper case (@, A-Z, left bracket, separator, right bracket, carat, and cursor). This is the software routine that causes the bizarre 'inverted' shift pattern on the TRS-80 . . . no shift for upper case!

If the character test at 0412 returns a value greater than or equal to 60, then 70 is subtracted (0429). No carry is generated if the test value was greater than or equal to 70, so this further separates the keyboard. See the diagram below:

```

.....
. 60 to 6F minus 70  0 1 2 3 4 5 6 7 .
. carries; result    B 9 :* ;+ ,< = > /? .
. is F0 to FF .
.....
. 70 to 77 minus 70 .
. does not carry;   ENT CLR BRK UPA DNA LFA RTA SPC .
. result is 00 to 07 .
.....

```

At address 043D, the value in the accumulator (00 to 07) is rotated left, producing the even values from 00 to 0E. The SHIFT byte in B is rotated right into the carry flag; if a carry is generated, the accumulator value is incremented (0442), providing the values 0+1, 2+1, 4+1 and so on – in other words, the odd values from 01 to 0F.

What follows is a classic example of machine language table look-up. HL is set to 0050, the address of the table in ROM; BC will be used as an offset, with B set to 0 and C made equal to A. When BC is added to HL, a resultant address (0050 to 005F) is produced, and the contents of that address is loaded up by the accumulator. Here is a look:

0443	21 50 00	LD	HL,0050
0446	4F	LD	C,A
0447	06 00	LD	B,0
0449	09	ADD	HL,BC
044A	7E	LD	A,(HL)
044B	57	LD	0,A

What do we find at 0050 to 005F? ASCII control codes. That result is stored in the D register (044B) before the termination sequence.

Table 2 - 9

Address	Contents	TRS-80 Action	ASCII Description	Keyboard Entry
0050	00	Carriage Ret.	Carriage Ret.	ENTER
0051	00	Carriage Ret.	Carriage Ret.	SHIFT ENTER
0052	1F	Clear Screen	Unit Separator	CLEAR
0053	1F	Clear Screen	Unit Separator	SHIFT CLEAR
0054	01	Break	Start of Heading	BREAK
0055	01	Break	Start of Heading	SHIFT BREAK
0056	5B	Up Arrow	Left Bracket	Up Arrow
0057	1B	Exit Escape	Escape	SHIFT Up Arrow
0058	0A	Line Feed	Line Feed	Down Arrow
0059	1A [00]	*See text	Substitute	SHIFT Down Arrow
005A	0B	Backspace	Backspace	Left Arrow
005B	1B	Backspace Line	Cancel	SHIFT Left Arrow
005C	09	Horizontal Tab	Horizontal Tab	Right Arrow
005D	19	32-Chr. Mode	End of Medium	SHIFT Right Arrow
005E	20	Space	Space	Space
005F	20	Space	Space	SHIFT Space
00100	; #####			
00110	; SIMPLE ACTIVE KEYBOARD DISPLAY ROUTINE TO SHOW THE USE			
00120	; AND PHANTOMING OF KEYS AS THEY ARE PRESSED BY THE USER			
00130	;			
00140	; DENNIS BATHORY KITSZ, ROXBURY, VERMONT 05669			
00150	; #####			
00160	;			
00170	ORG	5000H		
00180	;			
00180	; #####			
00200	; CLEAR SCREEN, DISABLE INTERRUPTS, CLEAR ACCUMULATOR,			
00210	; GET "0" CHARACTER, SCREEN POSITIONS, SCREEN OFFSET,			
00220	; AND BEGINNING OF KEYBOARD FOR TEST; ALL ARE SET UP ONCE			
00230	; #####			
00240	;			
00250	ENTER	CALL	01C9H	; CLEAR SCREEN TO START
00260	OI			; KILL ALL THEM BOTHERS
00270	XOR	A		; CLEAR ALL FLAGS
00280	START	LD	C,30H	; THIS IS THE "0" CHAR.
00290	LD	HL,3010H		; ADDRESS NEAR SCREEN CTR
00300	LD	0E,0024H		; DISTANCE BETWEEN LINES
00310	PUSH	0E		; SAVE LINE ADDRESS VALUE
00320	LD	0E,3B01H		; DEFINE FIRST KEYBOARD
00330	LD	B,0BH		; NUMBER OF LOOPS TO DO
00340	;			
00350	; #####			
00360	; TEST EACH ROW OF THE KEYBOARD, COLUMN BY COLUMN			
00370	; #####			
00380	;			
00390	LOOP	LD	A,(0E)	; FIRST ROW OF KEYBOARD
00400	BIT	0,A		; CHECK FIRST KEY COLUMN

Alright, we have upper case ASCII and TRS-80 control functions. How about the rest? Back up now to the test for SHIFT, at 0416. If such a shift is present, the value in A (40 to 5F) is increased by 20 (60 to 7F). These are the ASCII codes for lower case (@, a-z, left brace, separator, right brace, delete). The resultant code, as usual, is saved in the D register.

But what follows is curious:

0410	3A 40 3B	LD	A,(3B40)
0420	E6 10	AND	10
0422	2B 2B	JR	Z,044C

The keyboard is tested again, this time at row 3840, data position 10 – the down arrow. If that key is not depressed, the program skitters right to the termination routine at 044C, with the lower case ASCII code ensconced in the D register.

Why the SHIFT/down arrow combination? If the down arrow is depressed, the value in D is retrieved and placed in the accumulator (60 to 7F), then reduced by 60, becoming . . . aha! . . . 00 to 1F. The program jumps to the end sequence, with the accumulator clutching one of the complete set of 32 ASCII control codes!

(There is an anomaly in earlier Level II ROMs: the code for the down arrow at 0059 is returned before the control code. Later ROMs placed a 00 at 0059, resulting only in the return of a control code if SHIFT/down arrow was depressed.)

So where are we now? Upper and lower case, TRS-80 and ASCII control codes. We need numbers and figures, and so we shall have them. Recall the second diagram: at 042B, the command row was separated from the numbers, which were left at F0 to FF. At 042D, 40 is added, resulting in possible values of 30 to 3F. A further separation is made via a comparison with 3C:

30 to 3B	0	1	2	3	4	5	6	7	.
(CP 3C result)	B	9	:	;	<	=	.	>	/?

If the comparison is less than 3C, a carry is generated. The usual SHIFT test is made (at 0435), and if it fails, the value in A (30 to 3B) is maintained as the program moves into the end routine. These are the ASCII codes for numbers 0 to 9, colon and semicolon.

If the test value is 3C, 3D, 3E or 3F, no carry would be generated at 042F, and these values are XORed with 10. This toggles the high nibble

```

5016 71      00410      LD      [HL],C      ; FIRST DISPLAY A "0"
5017 2801    00420      JR      Z,JUMP1      ; DON'T CHANGE IF NO KEY
5019 34      00430      INC      [HL]        ; MAKE IT A "1" IF A KEY
501A 23      00440      JUMP1    INC      HL      ; NEXT SCREEN LOCATION
501B 23      00450      INC      HL      ; ...PLUS TWO
501C 23      00460      INC      HL      ; ...PLUS THREE
501D 23      00470      INC      HL      ; ...PLUS FOUR
501E CB4F    00480      BIT      1,A        ; SECOND KEYBOARD COLUMN
5020 71      00490      LD      [HL],C      ; FIRST DISPLAY A "0"
5021 2801    00500      JR      Z,JUMP2      ; DON'T CHANGE IF NO KEY
5023 34      00510      INC      [HL]        ; MAKE IT A "1" IF A KEY
5024 23      00520      JUMP2    INC      HL      ; NEXT SCREEN LOCATION
5025 23      00530      INC      HL      ; ...PLUS TWO
5026 23      00540      INC      HL      ; ...PLUS THREE
5027 23      00550      INC      HL      ; ...PLUS FOUR
5028 CB57    00560      BIT      2,A        ; THIRD KEYBOARD COLUMN
502A 71      00570      LD      [HL],C      ; FIRST DISPLAY A "0"
502B 2801    00580      JR      Z,JUMP3      ; DON'T CHANGE IF NO KEY
502D 34      00590      INC      [HL]        ; MAKE IT A "1" IF A KEY
502E 23      00600      JUMP3    INC      HL      ; NEXT SCREEN LOCATION
502F 23      00610      INC      HL      ; ...PLUS TWO
5030 23      00620      INC      HL      ; ...PLUS THREE
5031 23      00630      INC      HL      ; ...PLUS FOUR
5032 CB5F    00640      BIT      3,A        ; FOURTH KEYBOARD COLUMN
5034 71      00650      LD      [HL],C      ; FIRST DISPLAY A "0"
5035 2801    00660      JR      Z,JUMP4      ; DON'T CHANGE IF NO KEY
5037 34      00670      INC      [HL]        ; MAKE IT A "1" IF A KEY
5038 23      00680      JUMP4    INC      HL      ; NEXT SCREEN LOCATION
5039 23      00690      INC      HL      ; ...PLUS TWO
503A 23      00700      INC      HL      ; ...PLUS THREE
503B 23      00710      INC      HL      ; ...PLUS FOUR
503C CB67    00720      BIT      4,A        ; FIFTH KEYBOARD COLUMN
503E 71      00730      LD      [HL],C      ; FIRST DISPLAY A "0"
503F 2801    00740      JR      Z,JUMP5      ; DON'T CHANGE IF NO KEY
5041 34      00750      INC      [HL]        ; MAKE IT A "1" IF A KEY
5042 23      00760      JUMP5    INC      HL      ; NEXT SCREEN LOCATION
5043 23      00770      INC      HL      ; ...PLUS TWO
5044 23      00780      INC      HL      ; ...PLUS THREE
5045 23      00790      INC      HL      ; ...PLUS FOUR
5046 CB6F    00800      BIT      5,A        ; SIXTH KEYBOARD COLUMN
5048 71      00810      LD      [HL],C      ; FIRST DISPLAY A "0"
5049 2801    00820      JR      Z,JUMP6      ; DON'T CHANGE IF NO KEY
504B 34      00830      INC      [HL]        ; MAKE IT A "1" IF A KEY
504C 23      00840      JUMP6    INC      HL      ; NEXT SCREEN LOCATION
504D 23      00850      INC      HL      ; ...PLUS TWO
504E 23      00860      INC      HL      ; ...PLUS THREE
504F 23      00870      INC      HL      ; ...PLUS FOUR
5050 CB77    00880      BIT      6,A        ; SEVENTH KEYBOARD COLUMN
5052 71      00890      LD      [HL],C      ; FIRST DISPLAY A "0"
5053 2801    00900      JR      Z,JUMP7      ; DON'T CHANGE IF NO KEY
5055 34      00910      INC      [HL]        ; MAKE IT A "1" IF A KEY
5056 23      00920      JUMP7    INC      HL      ; NEXT SCREEN LOCATION
5057 23      00930      INC      HL      ; ...PLUS TWO
5058 23      00940      INC      HL      ; ...PLUS THREE
5059 23      00950      INC      HL      ; ...PLUS FOUR
505A CB7F    00960      BIT      7,A        ; EIGHTH KEYBOARD COLUMN
505C 71      00970      LD      [HL],C      ; FIRST DISPLAY A "0"
505D 2801    00980      JR      Z,JUMP8      ; DON'T CHANGE IF NO KEY
505F 34      00990      INC      [HL]        ; MAKE IT A "1" IF A KEY
5060 05      01000      JUMP8    PUSH     DE      ; SAVE THIS VALUE
5061 00E1    01010      POP      IX      ; PUT IT IN IX FOR A BIT
5063 01      01020      POP      DE      ; GET ORIGINAL DE VALUE
5064 19      01030      ADD      HL,DE      ; NOW START NEXT LINE
5065 05      01040      PUSH     DE      ; SAVE SAME VALUE AGAIN
5066 00E5    01050      PUSH     IX      ; STASH IT BRIEFLY
5068 01      01060      POP      DE      ; AND BACK INTO DE INTACT
5069 CB13    01070      RL      E        ; DEFINE NEXT KEYBOARD ROW
506B 10A6    01080      DJNZ     LOOP      ; DO IT FOR EIGHT ROWS
                    01090      ;
                    01100      ;
                    01110      ; CLEAR UP POINTERS AND DELAY SO SCREEN DOES NOT JITTER
                    01120      ;
                    01130      ;
5060 01      01140      POP      DE      ; CLEAR THE STACK
506E 010008  01150      LD      BC,800H      ; DELAY VALUE
5071 C06000  01160      CALL    0060H      ; DELAY SUBROUTINE IN ROM
5074 18BF    01170      JR      START      ; START THE ROUTINE AGAIN
                    01180      ;
                    01190      ;
5000 01200    01200      END      ENTER      ; BEGIN IT ALL HERE
00000 TOTAL ERRORS
31044 TEXT AREA BYTES LEFT

ENTER 5000 00250 01200
JUMP1 501A 00440 00420
JUMP2 5024 00520 00500
JUMP3 502E 00600 00580
JUMP4 5038 00680 00660
JUMP5 5042 00760 00740
JUMP6 504C 00840 00820
JUMP7 5056 00920 00900
JUMP8 5060 01000 00980
LOOP 5013 00390 01080
START 5005 00280 01170

```

from 3 to 2, resulting in values from 2C to 2F (, - . /). If a shift key was noted at 0437, the same toggle procedure is followed, changing values 30 to 3B into 20 to 2B (these would become space ! " # \$ ( ) \* = etc.).

0420	C6 40	A00	A,40
042F	FE 3C	CP	3C
0431	38 02	JR	C,0435
0433	EE 10	XOR	10
0435	CB 08	RRC	8
0437	30 12	JR	NC,0448
0439	EE 10	XOR	10
043B	1B 0E	JR	0448

Thus, the coding is complete: control codes (00 to 1F), punctuation (20 to 2F), numbers and figures (30 to 3F), upper case (40 to 5F) and lower case (60 to 7F). Just as an aside, the terms lower and upper case are sometimes written small and large case; old-time printers would chuckle at that. The case referred to is a printers case, which, when two were stacked one above the other, contained the capital and small letters. Thought you might like to know that.

Back to the routine, starting at the termination sequence (044C); the decoded character is saved in D, and that is the only information we need to preserve, since the bulk of the work is done.

044C	01 AC 00	LD	BC,00AC
044F	C0 60 00	CALL	0060
0452	7A	LD	A,D

A delay at 0060 is called, which was intended to wait through the bounce present with normal mechanical switch contacts – but the easily dirtied switches on the TRS-80 are abnormal! This delay uses the accumulator, and when it is free, the value in D is restored to it. This value is compared to 01 (the BREAK code), and returns directly to the main routine (0455) with any code other than BREAK.

If BREAK is discovered, the program executes a call to 0028 (RST 28), returning to Level II.

The routine is quite efficient, and is capable of returning 128 different values at a rate of better than 100 per second – ten times the speed of the world's fastest typist!



## Software Modifications

Software makes the computer. With that in mind, it's not hard to understand the popularity of the TRS-80. Its BASIC is simple to use and immediately accessible. It reports back errors and provides clear screens and graphics with easy commands. At first, it was hard to imagine how such an elegant BASIC could be improved.

Such illusions could not last long, especially when weakly designed hardware started to exhibit keybounce, when machine language software was so fast but so difficult to access, when putting a program aside meant losing all variables, and so forth.

In this Chapter, several simple but very important software modifications will be presented :

Keyboard debounce with repeating keys, audible beep tone, and an upper/lower case driver.

Two methods of intercepting the BASIC interpreter in order to create your own commands.

Packing machine language programs in simple BASIC strings, where they can be moved about and accessed easily.

Sound and sound-effects generation routines.

Creating somewhat unlistable BASIC programs.

Auto-execution of SYSTEM programs, including an auto-load BASIC module.

A simple machine language monitor accessible directly from BASIC.

BASIC is not an incomprehensible, immutable, indivisible whole, but rather a pliant, carefully sewn, patchwork quilt of useful subroutines. These routines are accessed singly or as groups, not only whenever one of this BASIC's English-language commands is entered, but even while waiting for commands to be entered or programs to be run.

Chapter 1 contained an overview of the structures that BASIC is composed of; below is a more detailed look at the building blocks out of which Level II is created.

1. A power-up sequence: including preparation of blocks of reserved memory, clearing of internal hardware systems, and total memory examination.
2. Numeric conversion routines: single precision to integer and vice versa, numeric to string and vice versa, assignment of levels of precision.
3. Simple arithmetic operations: addition, subtraction, multiplication, division, comparison, and raising to a power (exponentiation); integer, single, and double-precision calculations.

4. Mathematical functions: activities based on the sine function, as well as logarithms, square roots, absolute value and truncation, and random number generation.

5. String operations: concatenation and truncation of strings, direct keyboard scan conversions (INKEY\$), alphabetic comparisons.

6. Variables: assignment of numeric, string, and array variable names, variable assignments (LET), updating of values, searching for variable names. Partly integrated with NEW routines.

7. Keyboard input: polling of the keyboard matrix, conversion to characters, searching for carriage return, building a keyboard buffer.

8. Cassette input/output: motor relay control, assembly of parallel data into serial form, timing of output pulses, timing of input, reassembly of data from serial to parallel form.

9. Video input/output and display management: screen clear, scrolling, tabbing, character display including line feed, carriage return, backspace, set/reset, POS, POINT, cursor control, characters per line.

10. Printer control: lines per page, top of form, output of characters, waiting for handshake.

11. A command interpreter for organizing the entry points and order of chosen subroutines.

12. Error reporting routines.

13. Program line management routines. Partly integrated with NEW routines.

14. Editing functions: Insert, delete, kill, exit, etc. Integrated in part with program line management routines.

15. Run-time management: Integrated with most of the above functions, but including subroutine handling, loop handling, etc.

A more complete rundown on the TRS-80 Level II ROM memory map can be found in *Inside Level II, Supermap, Microsoft BASIC Decoded*, and *TRS-80 Disassembled Handbook* (see Appendix II for details).

Listing 3-1. Custom BASIC interpreter patch routine.

```

00100 ; #####
00110 ; DEBOUNCE, AUTO-REPEAT, AND BEEP KEYBOARD PATCH ROUTINE.
00120 ; THIS ROUTINE AS WRITTEN PATCHES INTO 4016. TO USE
00130 ; OTHER DRIVERS SUCH AS LOWERCASE, GRAPHICS, ETC., THE
00140 ; RETURN ADDRESS (NORMALLY D3E3) MUST BE REPLACED BY THE
00150 ; ADDRESS OF THE OTHER DRIVER. ALSO, TO USE THIS ROUTINE
00160 ; WITH THE CUSTOM INTERPRETER /ON AND /OUT COMMANDS BY
00170 ; PATCHING THIS DRIVER IN PLACE WITH /ON AND THE NORMAL
00180 ; (OR UPPER/LOWER, ETC.) DRIVER IN PLACE WITH /OUT.
00190 ; #####
00200 ;
4016 0030 00210 ORG 4016H ; KEYBOARD SCAN PATCH
00220 DEFW ENTREE ; START OF DEBOUNCE ROUT.
00230 ORG 3000H ; NOTE THAT THIS UTILITY
00240 ; IS CURRENTLY SET UP FOR
00250 ; USE WITH A MEMORY ADD'N
00260 ; AT 3000H. IT CAN BE
00270 ; RE-ORGED AT ANY LOC'N.
06CC 00280 BASIC2 EQU 06CCH ; BASIC "READY" DISPLAY
4036 00290 KEYHLD EQU 4036H ; NORMAL KEYSTROKE STORE
3801 00300 KEYBRD EQU 3801H ; FIRST KEYBOARD ADDRESS
401A 00310 HOLDER EQU 401AH ; RESERVED BYTE FOR DELAY
4099 00320 INKEYS EQU 4099H ; INKEY$ STORAGE BYTE
0060 00330 DELAYS EQU 0060H ; ROM DELAY SUBROUTINE
00340 ;
00350 ; #####
00360 ; ROUTINE BEGINS HERE, PATCHING ITSELF INTO PLACE AT 4016
00370 ; #####
00380 ;
3000 E5 00390 ENTREE PUSH HL ; SAVE HL IF TRANSPARENT
3001 210830 00400 LO HL,START ; GET START OF ROUTINE
3004 221640 00410 LO [4016H],HL ; PUT INTO KEYBOARD PATCH
3007 E1 00420 PDP HL ; RESTORE HL VALUE
3008 C3CC06 00430 JP BASIC2 ; AND GO BACK TO "READY"
00440 ;
00450 ; #####
00460 ; THIS AREA MAKES THE PRELIMINARY CHECK OF KEYBOARD ROWS
00470 ; #####
00480 ;
3008 213640 00490 START LO HL,KEYHLD ; SET UP STORAGE AREA
300E 010138 00500 LO BC,KEYBRD ; SET UP FIRST KBD ROW
3011 1600 00510 LO 0,0 ; SET UP COUNTER OF ROWS
3013 0A 00520 CHKKEY LO A,(BC) ; FIND IF A KEY PRESSED
3014 5F 00530 LO E,A ; SAVE VALUE IN E REG.
3016 A3 00540 AND E ; TEST IF KEY WAS PRESSED
3016 2018 00550 JR NZ,CKPREV ; IF YES, SEE IF SAME ONE
3018 77 00560 LO (HL),A ; SAVE VALUE IN STORAGE
3019 14 00570 INCO INC 0 ; INCREMENT ROW COUNTER
301A 2C 00580 INC L ; INCREMENT STORAGE AREA
3018 C801 00580 RLC C ; SHIFT TO NEXT KBD ROW
3010 79 00600 LO A,C ; GET VALUE OF KBD ROW
301E 0680 00610 SUB B0H ; CHECK IF SHIFT KEY ROW
3020 20F1 00620 JR NZ,CHKKEY ; IF NOT THEN CONTINUE
00630 ;
00640 ; #####

```

Listing Continued . . .

## Sophisticated Debouncing

The first few hundred thousand TRS-80's were afflicted with serious keybounce problems—the appearance of double letters when only a single letter was typed. Full-scale preventive maintenance is presented in the following chapter; but there are software solutions as well. If debounce were the only criterion, though, maintenance would be the ideal solution.

But the software designers made no provision for repeating keys, nor did the hardware designers include access to lower case characters. Furthermore, the silent keyboard remains a frustration to touch-typists and others who do not refer constantly to the screen for feedback.

Thus, some sort of audible reinforcement (as with the Apple's entry-error beep) would be a thoughtful addition to the keying process.

Listing 3-1 is a complete debounce, audible beep, key repeat, and upper/lower case driver routine. The program is written as three independent subroutines, each of which may be disabled or removed before assembly.

## Continued Listing

```

00650 ; CHECKING IS DONE - NOW SEE IF PREVIOUS KEYS HELD DOWN
00660 ; #####
00670 ;
3022 0607 00680 LD B,7 ; ELSE GET NUMBER OF ROWS
3024 20 00690 DECL DEC L ; MOVE BACK THRU STORAGE
3025 86 00700 ADD A,[HL] ; AND MAKE TOTAL OF KEYS
3026 10FC 00710 DJNZ DECL ; AND DO IT FOR ALL ROWS
3028 A7 00720 AND A ; TEST IF ANY KEYS STORED
3028 3E0D 00730 LD A,0 ; A=0, FLAGS REMAIN SAME
3028 C0 00740 RET NZ ; BACK IF KEY IN STORAGE
302C 321A40 00750 LD (HOLDER),A ; SAVE NEW VALUE IN CTR.
302F C9 00760 RET ; AND BACK TO MAIN ROUT.
00770 ;
00780 ; #####
00790 ; NEXT TEST IS FOR STATUS OF INKEY$, IF IT IS IN USE
00800 ; #####
00810 ;
3030 A6 00820 CKPREV AND [HL] ; SEE IF VALUE IS SAME
3031 2B1F 00830 JR Z,STORE ; STORE VALUE IF ZERO
3033 3AB840 00840 LD A,[INKEY$] ; FIND VALUE AT INKEY$
3036 A7 00850 AND A ; SEE IF SOMETHING THERE
3037 20E0 00860 JR NZ,INCO ; IF SO THEN GO AWAY
3039 3A1A40 00870 LD A,(HOLDER) ; GET DELAY COUNTER VALUE
303C 3C 00880 INC A ; INCREMENT THE COUNTER
303D 321A40 00890 LD (HOLDER),A ; AND SAVE VALUE BACK
3040 FEFF 00900 CP OFFH ; IS COUNTER AT END YET?
3042 2B09 00910 JR Z,DECA ; IF SO, THEN REPEAT
00920 ;
00930 ; #####
00940 ; REPEATING-KEY TIME-WASTE VALUE (FF) MAY BE VARIED
00950 ; #####
00960 ;
3044 C5 00970 PUSH BC ; SAVE BC FOR LATER
3045 06FF 00980 LD B,OFFH ; GET DELAY VALUE
3047 00 00990 NOP ; WASTE SOME TIME
3048 10FD 01000 DJNZ TMWSTE ; AND DO IT FF TIMES
304A C1 01010 POP BC ; RESTORE BC VALUE
3048 18CC 01020 JR INCO ; AND GO BACK TO SCANNING
304D 30 01030 DECA DEC A ; MAKE A BECOME FE
304E 321A40 01040 LD (HOLDER),A ; AND SAVE IT IN DELAY
3051 78 01050 LD A,E ; GET KEYSTROKE FOUND
3052 73 01060 STORE LD (HL),E ; AND PUT IT IN STORAGE
01070 ;
01080 ; #####
01090 ; DEBOUNCE BELOW MAY BE ELIMINATED BECAUSE BEEP USES TIME
01100 ; #####
01110 ;
3053 C5 01120 PUSH BC ; SAVE VALUE IN BC
3054 0100D2 01130 LD BC,200H ; GET DEBOUNCE DELAY
3057 C06000 01140 CALL DELAYS ; AND CALL ROM DELAY
305A C1 01150 POP BC ; AND GET VALUE TO BC
305B 0A 01160 LD A,[BC] ; GET VALUE AT KEYBOARD
305C A3 01170 AND E ; AND TEST IF IT'S THERE
305D C8 01180 RET Z ; IF NOT IT WAS BOUNCE
01190 ;
01200 ; #####
01210 ; BEEP ROUTINE PRODUCES VERY SOFT (NOT ANNOYING) SOUND
01220 ; #####
01230 ;
305E C5 01240 PUSH BC ; ELSE SAVE THE VALUE
305F E5 01250 PUSH HL ; AND SAVE THE LOCATION
3060 F5 01260 PUSH AF ; AND SAVE THE KEYSTROKE
3061 0640 01270 LD B,40H ; AND KEY BEEP DURATION
3063 3A3040 01280 LD A,[4030H] ; AND GET SCREEN STATUS
3068 E6F0 01290 AND OFDH ; AND MASK OUT ALL BITS
3068 67 01300 LD H,A ; SAVE WAVE "0" MASK IN H
3069 F602 01310 OR 2 ; CREATE A WAVE "1" MASK
3068 8F 01320 LD L,A ; SAVE WAVE "1" MASK IN L
306C 70 01330 LD A,L ; GET THE WAVE "1" MASK
306D 03FF 01340 OUT (OFFH),A ; AND CREATE WAVEFORM 1
306F 7C 01350 LD A,H ; GET THE WAVE "0" MASK
3070 03FF 01360 OUT (OFFH),A ; AND CREATE WAVEFORM 0
3072 C5 01370 PUSH BC ; SAVE THE DURATION VALUE
3073 0640 01380 LD B,40H ; GET THE PITCH VALUE
3075 10FE 01390 DJNZ $+0 ; AND WAIT THRU WAVEFORM
3077 C1 01400 POP BC ; RESTORE THE DURATION
3078 10F2 01410 DJNZ LOOP ; AND DO FOR FULL BEEP
307A F1 01420 POP AF ; RESTORE KEYSTROKE VALUE
3078 E1 01430 POP HL ; RESTORE STORAGE VALUE
307C C1 01440 POP BC ; RESTORE COUNTER VALUE
307D C3F803 01450 JP 03F8H ; AND RETURN TO KEYSKAN
01460 ;
01470 ; #####
01480 ; #####
01480 END BASIC2
BASIC2 06CC 00280 00430 01480
CHKKEY 3D13 00520 00620
CKPREV 303D 00620 00550
DECA 304D 01030 00910
DECL 3024 00690 00710
DELAYS 006D 00330 01140
ENTREE 300D 00390 00220
HOLDER 401A 00310 00750 00870 00890 01040
INCO 3019 00570 00860 01020
INKEY$ 4099 00320 00840
KEYBR0 3601 00300 00500
KEYHLD 4036 00290 00490
LOOP 306C 01330 01410
START 3D08 00490 00400
STORE 3052 01060 00830
TMWSTE 3047 00990 01000

```

This routine patches into the keyboard control block driver address at 4016, leading the program to its own entry point instead of 03E3 (see the Supplement to Chapter 2 for details on the operation of the TRS keyboard scan).

At START, the keyboard scan begins with HL pointing to the first position in a keystroke storage buffer (4036). BC points to the first keyboard row (3801). The program proceeds similarly to the normal Level II scan, except that a location (401A) has been set aside to 'count down' the time a key remains pressed. If any key or combination of keys remain pressed for the duration of the loop, the character (normally rejected by Level II's rollover capabilities) is accepted again. This is the start of the repeating process. The INKEY\$ storage area is checked (so that programs using INKEY\$ are not delayed by an unusable character acceptance), and a short debounce-delay loop is entered.

If a legitimate key is found, a different debounce-delay loop is entered, and a rapidly-fluctuating one-zero pattern is sent to the cassette port (FF). This sounds as a beep if an amplifier and speaker are connected. The routine can be exited before the beep, so only the debounce-repeat options are present; it can also be exited after the beep, returning to the main routine with the keystroke.

The final portion of the program is an upper/lower case driver program. This driver is irrelevant unless you have a lower case modification in place, and should be disabled (or not assembled) if you have not made the modification. It merely strips the conversion to upper case normally made by the Level II software, and returns to the running program with the actual key depressed instead of an upper case converted version.

## Upper/Lower Case Driver

When the TRS-80 keyboard is used, all characters are automatically converted to upper case before being displayed. The keyboard itself, however, returns a full upper/lower case value (albeit inverted - shift for lower case) to the display routine. The display routine then sends this information to the screen. The screen always displays upper case because the hardware to provide lower case was not a part of the TRS-80 as sold. The addition of a single integrated circuit (see Chapter 4) provides this access.

The lower-case driver patches into the display routine just as the character to be displayed is returned in the accumulator. Control is taken

```

00100 ; #####
00110 ; SIMPLE LOWER-CASE DRIVER ROUTINE FOR BASIC AND DOS
00120 ; #####
00130 ;
7F00 00140      ORG      07F00H      ; NEAR TOP OF BASIC
00150 ;
00160 ; #####
00170 ; GET CONTROL BLOCK FOR SCREEN AND CHANGE CHARACTERS
00180 ; #####
00180 ;
7F00 006E03 00200 LCDRIV LD      L,(IX+3)      ; IX POINTS TO DEVICE
7F03 006B04 00210      LO      H,(IX+4)      ; CONTROL BLOCK (VIDEO)
7F06 0A9A04 00220      JP      C,049AH      ; BACK TO SCREEN DRIVER
7F09 007E05 00230      LO      A,(IX+5)      ; GET CURSOR CHARACTER
7F0C 87      00240      DR      A          ; CHECK IF CURSOR IS ON
7F00 2801    00250      JR      Z,GETCHR     ; GET ONE IF CURSOR OFF
7F0F 77      00260      LO      [HL],A      ; PUT CURSOR INTO POSN.
7F10 78      00270 GETCHR LD      A,C      ; GET CHARACTER TO SHOW
7F11 FE20    00280      CP      20H        ; SEE IF A CONTROL CODE
7F13 0AD605 00280      JP      C,0506H      ; BACK TO DRIVER IF C.C.
7F16 FE80    00300      CP      80H        ; SEE IF GRAPHIC CHAR.
7F18 02A604 00310      JP      NC,04A6H      ; BACK TO DRIVER IF SO
7F1B FE58    00320      CP      05BH      ; CHECK UPPER/LOWER CASE
7F10 3008    00330      JR      NC,CHECK1    ; IF >5B, CHECK FURTHER
7F1F FE40    00340      CP      40H        ; CHECK UPPER CASE
7F21 380E    00350      JR      C,GDAWAY     ; IF <40, CHECK NO FURTHER
7F23 C620    00360      ADD     A,20H      ; IF 40-5B, MAKE UPPER
7F25 180A    00370      JR      GDAWAY     ; DONE - BACK TO DRIVER
7F27 FE78    00380 CHECK1 CP      7BH      ; SEE IF ALPHABETIC
7F28 3D06    00380      JR      NC,GDAWAY     ; NO FURTHER IF NOT ALPHA
7F2B FE60    00400      CP      60H        ; SEE IF ALPHABETIC
7F2D 3802    00410      JR      C,GDAWAY     ; NO FURTHER IF NOT ALPHA
7F2F 0620    00420      SUB     20H        ; PLAY SWITCH TO LOWER
7F31 C37D04 00430 GDAWAY JP      0470H      ; OUT TO DRIVER NOW
00440 ;
00450 ; #####
00460 ; PUT VIDEO PATCH INTO PLACE UPON LOAD
00470 ; #####
00480 ;
401E 00490      ORG      401EH      ; THIS IS VIDEO PATCH
401E 007F    00500      DEFW     LCDRIV      ; PUT LCDRIV ROUTINE IN
00610 ;
00520 ; #####
06CC 00530      END      06CCH      ; BACK TO BASIC READY
00000 TOTAL ERRORS

```

Listing 3-13. Upper/lower case driver.

## Using The Editor/Assembler

The Editor/Assembler is one of the most powerful tools available to the TRS-80 customizer. It is a fast, high-level compiler which produces a block of Z-80 machine code. Its job is to provide an easily accessible substitute for the tedious creation of bytes of Z-80 coded information.

The Z-80 microprocessor is capable of responding to many hundreds of combinations of ones and zeros. Each pattern causes the Z-80 to follow a unique pattern of electronic activity, and many thousands of those activities in concert create a sophisticated language like BASIC.

Using these patterns can be very tricky and time-consuming. Long ago, computer designers learned that it was easier to remember an action like 'load the accumulator with the contents of byte counter register' as 'Load A with B', abbreviated LD A,B. This is much handier than trying to recall 01111000. These abbreviations are called mnemonics, which are what you will find in all the program listings in this book.

You will also find that, instead of specific locations in memory (such as 3C00), there may

from the convert-to-upper-case display function in ROM. Ideally, this ROM routine could just be entered after its convert-to-upper case code; unfortunately, this would result in the famous inverted display . . . normal upper case, shifted lower case.

To avoid this, the character is tested and converted to its proper case before being returned to the ongoing display driver routine in ROM. Notice something interesting: when programs are listed with this driver, the letters appear in lower case. That is because when the programs are entered, they are in fact being entered with the keyboard *unshifted*. Because this can be a bit disconcerting (and also quite illegible, since we all are used to upper case lists), an upper case on/off software patch is provided.

be a 'label' (such as VIDEO). Once VIDEO has been defined as 3C00 to the Editor/Assembler, it will always interpret the label as the number that was assigned to that label.

Line numbers are provided to keep things in order and to insert or edit pieces of code, and there is space on every line for comments.

Load the Editor/Assembler tape under the SYSTEM command. Its name is EDTASM. When the loading is complete, enter a slash (/), and you will be presented with the prompt:

```

TRS-80 EDITOR/ASSEMBLER 1.1
* _

```

This is EDTASM's equivalent of the BASIC prompt:

```

RADIO SHACK LEVEL II BASIC
READY
> _

```

You are being asked for input. Unlike BASIC, EDTASM has only a few commands. They are (in the order you are likely to use them):

**I**

This command inserts numbered program lines almost exactly like the BASIC command AUTO. When I is entered alone, numbering starts with line 100 in increments of 10 line numbers. On the other hand, I15,15 will start with line 15 in increments of 15.

**P**

The equivalent of a list. A single P lists the next sixteen lines of the program. P10:100 lists 10 to 100. P# is the first line, P. is the current line, and P\* is the last line.

**N**

Here is the renumber command. All lines are

automatically renumbered in increments of 10 starting with line 100. Again, specific lines and increments may be specified: N300,50 will renumber all lines in increments of 50, with the first line being 300.

### L

Loads a source tape, but not an object tape. Up to a six-character name may be specified.

### W

Writes a source tape (the program listing). Up to a six-character name may be specified.

### D

Deletes the specified line or lines. D# deletes the first line, D. deletes the current line, and D\* deletes the last line. Groups of lines are specified with a colon, as D40:170 or D#:90.

### E

The edit function. The pound (#), period (.), and asterisk (\*) represent the first, current, and last lines. A line number (as E400) may be specified. The editing functions are identical to BASIC's editing functions – except that characters to be deleted are *not* delimited by exclamation points.

### R

Replaces the indicated line. The line number is presented, and new information may be entered.

### F

This command finds a text string. It is not followed by a space. To search for the phrase 'ENTRY', type FENTRY (ENTER). The entire line containing the phrase will be printed. To find the next identical phrase, merely type F (ENTER).

### H

This sends the source listing, unassembled, to the printer. The complete source listing, including line numbers, is printed. As usual, (#), (.) and (\*) may be used to indicate first, current, and last line, and groups of lines may be printed (as H55:3000).

### T

The poor person's text editor. The source code is sent to the printer without line numbers. Thus, text may be entered a line at a time, and the numberless result printed. The same functions provided with H are available.

### B

The exit to BASIC. As sold, EDTASM returns only to MEMORY SIZE?, and all programs and information, including EDTASM itself, are lost. Patches are available to re-route this exit.

### A

This command directs EDTASM to compile your source code into object code, make a list of all the labels (symbols) used, and check for errors. The A command may be followed by a six-letter name, as well as the 'switches' /NL (no listing), /NS (no symbol table), /NO (no object code), or /WE (wait upon error). The switches may be used in any combination, and are useful in shaking the errors out of an assembly program.

Lines are always entered into EDTASM under the I (insert) or R (replace) commands. A line number is presented, so:

```
00010*_
```

Several columns are then available, consisting respectively of an optional label, the mnemonic instruction, the 'operand' (if any), and any comments (always following a semicolon). A complete group of lines would look like this:

```
00010 VIDEO EQU 3C00H ;SCREEN TOP
00020 ORG 5000H ;START PROGRAM
00030 ENTRY LD A,B ;GET B INTO A
00040 LD HL,VIDEO ;HL AT SCREEN
```

This excerpt gives this information: the label VIDEO is an 'equate' (is defined as) location 3C00. The program starts (has its origin – ORG) at 5000. The label ENTRY is assigned to the start of the program, and that program's first action is to load the accumulator with register B. Next, the HL register is pointed to VIDEO (3C00), the start of the screen memory.

When told to assemble this (using the A command), the results will look like the following:

```
3C00 00010 VIDEO EQU 3C00H ;SCREEN TOP
5000 00020 ORG 5000H ;START PROGRAM
5000 7B 00030 ENTRY LD A,B ;GET B INTO A
5001 21003C 00040 LD HL,VIDEO ;HL AT SCREEN
```

The EDTASM program evaluated all the information in the source code and created the columns at the left. The first column specifies the current address, and the second column specifies the machine language code, if any, for that particular line. Note the correct assignment of 3C00 to VIDEO in line 00040. 7B is the machine code for LD A,B and 21 is the machine code for LD HL,NNNN. In this case, NNNN is VIDEO is 3C00.

For detailed instructions, refer to the EDTASM instruction manual. One thing to note: you can conserve source code memory space by using the right arrow (tab) instead of spacing between program lines, labels, commands, operands, and comments. Each tab is a single character, but spaces are counted separately.

```

00100 ; #####
00110 ; CUSTOM BASIC INTERPRETER PATCH FOR USE WITH ALL OF THE
00120 ; ROUTINES PRESENTED IN CUSTOM TRS-80 WHICH ARE TRANS-
00130 ; PARENT TO BASIC. THESE ROUTINES ARE CALLED BY THIS
00140 ; ROUTINE, WHICH HAS ALREADY PLACED A RETURN VALUE ON THE
00150 ; STACK. ALL TRANSPARENT ROUTINES MUST EXECUTE A RETURN
00160 ; INSTRUCTION AS THEIR FINAL INSTRUCTION TO WORK WITH
00170 ; THIS CUSTOM INTERPRETER ROUTINE. THE COMMANDS AVAIL-
00180 ; ABLE WITH THIS INTERPRETER ARE:
00190 ; /LOAD /SAVE /NEW /OPEN
00200 ; /ON /OFF /GET /PUT
00210 ; /STEP /MEM
00220 ; AND OTHER USER-DEFINED "/" COMMANDS AND ROUTINES.
00230 ; THIS ROUTINE HAS PUSHED THE RETURN ADDRESS (1078) ON
00240 ; THE STACK. THE ROUTINE JUMPED TO IS A PSEUDO-CALL
00250 ; IN THAT IT EXECUTES A "RET", THUS RETURNING TO 1078.
00260 ; #####
00270 ;
1078 00280 BYTE EQU 1078M ;ROM READ KEYS & TOKENIZE
00290 ;
00300 ; #####
00310 ; CHECK THAT THE BASIC STACK IS IN INTERPRETATION MODE
00320 ; #####
00330 ;
0000 E3 00340 BEGIN EX (SP),ML ; GET SP INTO ML FOR TEST
0001 70 00350 LO A,L ; GET L INTO A FOR TEST
0002 FE58 00360 CP 58H ; IS LSB OF STACK 58?
0004 2003 00370 JR NZ,NOTROY ; NOT INTERPRETING IF NZ
0006 7C 00380 LO A,M ; GET H INTO A FOR TEST
0007 FE10 00380 CP 10H ; IS MSB OF STACK 10?
0008 E3 00400 NOTROY EX (SP),HL ; RESTORE STACK TO SP
000A C27810 00410 JP NZ,BYTE ; IF NOT 10H THEN TO ROM
00420 ;
00430 ; #####
00440 ; MUST HAVE BEEN AT 1058 FOR INTERPRETATION; THEREFORE,
00450 ; CHECK TO SEE IF SPECIAL SLASH (/) COMMAND THAT IS NEXT
00460 ; #####
00470 ;
0000 C07810 00480 CALL BYTE ; READ CHAR. & TOKENIZE
0010 F5 00490 PUSH AF ; SAVE VALUE READ
0011 FE00 00500 CP 0000H ; IS IT "/" COMMAND?
0013 2805 00510 JR Z,OKSLSH ; IF SO, THEN CONTINUE
0015 F1 00520 POP AF ; ELSE RESTORE AF VALUE
0015 28 00530 DEC HL ; PUT POINTER BACK ONE
0017 C37810 00540 JP 1078M ; AND BACK TO NORMAL ROM
00550 ;
00560 ; #####
00570 ; SLASH (/) HAS BEEN FOUND, THEREFORE MUST BE COMMAND
00580 ; #####
00590 ;
001A F1 00600 OKSLSH POP AF ; RESTORE VALUE TO AF
0018 C07810 00610 CALL BYTE ; NEXT COMMAND IN LINE
001E 2003 00620 JR NZ,SAVE ; GO IF ONE IS IN PLACE
0020 C38718 00630 SYNERR JP 1887M ; ?SN ERROR IF LINE END
00640 ;
00650 ; #####
00660 ; SINCE SLASH & NEXT CHARACTER HAS BEEN FOUND, NOW NEXT
00670 ; CHARACTER IN LINE MUST BE TESTED FOR VALIDITY AS USER-
00680 ; DEFINED COMMAND. SEE ABOVE FOR THOSE AVAILABLE HERE.
00690 ; #####
00700 ;
0023 117810 00710 SAVE LO DE,1078H ; GET RETURN ADDRESS
0028 05 00720 PUSH DE ; PLACE IT ON STACK
0027 FE40 00730 CP 0A0H ; ---- SAVE ----
0028 CA0000 00740 JP Z,SAVER ; GO TO SAVE ROUTINE
002C FE88 00750 CP 08BH ; ---- NEW ----
002E CA0000 00760 JP Z,RENEW ; GO TO RENEW ROUTINE
0031 FE42 00770 CP 0A2H ; ---- OPEN ----
0033 CA0000 00780 JP Z,OPENER ; GO TO OPEN ROUTINE
0036 FECC 00790 CP 0CCM ; ---- STEP ----
0038 CA0000 00800 JP Z,STPSET ; GO TO STEPPING ROUTINE
0038 FEC8 00810 CP 0C8H ; ---- MEM ----
0030 CA0000 00820 JP Z,MEMSET ; GO TO MEMORY SET ROUT.
0040 FEEA 00830 CP 0EAH ; ---- LOC ----
0042 CA0000 00840 JP Z,RELOC ; GO TO RELOCATION ROUT.
0045 FE41 00850 CP 0A1H ; ---- ON ----
0047 CA0000 00860 JP Z,KEYON ; GO TO DEBOUNCE ON ROUT.
004A FE40 00870 CP 0A0H ; ---- OUT ----
004C CA0000 00880 JP Z,KEYOFF ; KILL DEBOUNCE ROUTINE
004F FE44 00890 CP 0A4H ; ---- GET ----
0051 CA0000 00900 JP Z,COPYIN ; GO TO READ TAPE ROUTINE
0054 FE45 00910 CP 0A5H ; ---- PUT ----
0056 CA0000 00920 JP Z,DU88ER ; GO TO WRITE TAPE ROUT.
0058 C32000 00930 JP SYNERR ; ?SN ERROR IF UNDEFINED
00940 ;
00950 ; #####
00960 ; THIS ROUTINE WILL NOT ASSEMBLE AS IT STANDS. IT MUST
00970 ; BE APPENDED TO THE OTHER ROUTINES WHICH WILL BE USED
00980 ; IN CONJUNCTION WITH BASIC. ALL THE TERMS LISTED ABOVE
00990 ; MUST BE DEFINED, OR ELSE THEY MUST BE DELETED FROM THE
01000 ; ASSEMBLY LISTING.
01010 ; #####
01020 ;
0000 01030 END
00010 TOTAL ERRORS

```

## Patching the BASIC Interpreter

Each time a BASIC command is entered or a program line is being run, a section of ROM evaluates each of these commands in order, jumping to internal subroutines that will produce the desired result. This section of ROM is called the interpreter, an area which translates the commands into program action.

At address 4003, the machine language instruction C3 78 1D can be found, which means 'jump to address 1D78'. 1D78 is the main entry point to the BASIC interpreter. But the routine can be intercepted *before* going to 1D78, by patching a different jump into addresses 4004 and 4005.

This intercept is very important, because every BASIC - transparent software modification in this book will be patched into this location, leading to the master custom interpreter program below (Listing 3-2). When a command line is entered, the program in Listing 3-2 intercepts the routine at 4003, and first examines the status of the stack pointer; if it points to 1D5B, then the intercept program knows BASIC is in the interpretation mode.

Its next step is to CALL 1D78. By calling 1D78 instead of jumping to it, a 'tokenized' version of the next command in line is returned to the master custom interpreter. Tokenizing is an important, specialized process which allows the BASIC listings to use very little memory and allows the interpreter to evaluate commands at high speed.

When the token is returned to the custom interpreter, it can then be evaluated to see if it is a specially designated indicator command.

If the slash command indicator is found by the custom interpreter, it moves on to a lookup table to search for one of the specialized commands. All these commands will be tokens as well, so only a single byte comparison need be made.

There is another method of patching into the BASIC interpreter. If you are a Level II user, merely enter the command OPEN. Very promptly the computer will respond with '?L3 ERROR'. What is an '?L3 ERROR'? It refers to a 'Level III Error', the extended BASIC that is available as a part of the TRS-80 disk system.

Now enter the statement OPNE. This time a '?SN ERROR' is produced. How does the machine know that OPEN is a disk command and that OPNE is just garbage?

```

00100 ;
00110 ; #####
00120 ; FULL-FEATURED KEY/SCREEN DRIVER - DENNIS BATHORY KITSZ
00130 ; THIS ROUTINE IS A LEVEL II KEYBOARD REPLACEMENT ROUTINE
00140 ; CAPABLE OF PROVIDING: AUTOREPEAT AFTER SELECTED DELAY
00150 ; (FOUND IN 8 REGISTER IN DELAY SECTION); BEEP WITH ANY
00160 ; CHOICE OF PITCH; RESULTANT DEBOUNCE; CORRECTED SHIFT-
00170 ; DOWN ARROW CONTROL CODE FOR EARLIER LEVEL II ROMS; A
00180 ; SHIFT-0 SELECTABLE UPPER/LOWER CASE DRIVER AND DISPLAY.
00190 ; NOTE THAT THIS ROUTINE IS SET UP FOR USE AT 3039 HEX
00200 ; (12345 DECIMAL) FOR ENTRY IN THE MEMORY SIOECAR WHICH
00210 ; IS ADDRESSED FROM 3000 TO 3700 HEX. IT MAY BE SET TO
00220 ; ANY ORIGIN OF THE USER'S CHOICE, SUCH AS USUAL HIGH MEM
00230 ; #####
00240 ;
40B9 00250 INKEYS EQU 40B9H ;INKEY$ BYTE STORAGE AREA
4030 00260 PORTFF EQU 4030H ;CASSETTE OUTPUT PORT
401A 00270 KPLACE EQU 401AH ;1-BYTE KEYSTROKE STORE
4018 00280 SHIFTR EQU 4019H ;STORAGE FOR LC DRIVER
00290 ;
00300 ; #####
00310 ; PATCH KEYBOARD ROUTINE INTO 4016 AND DISPLAY INTO 401E
00320 ; #####
00330 ;
4016 00340 ORG 4016H ; START OF KEYBOARD SCAN
4016 3830 00350 DEFW KBPFIX ; PATCH KEYBOARD ROUTINE
401E 00360 ORG 401EH ; START OF DISPLAY SWEEP
401E 2A31 00370 DEFW LOWER ; PATCH UPPER/LOWER ROUTINE
00380 ;
3038 00390 ORG 3039H ; START AT MEMORY SIOECAR
00400 ;
00410 ; #####
00420 ; SET STORAGE #1, ROW #1, COUNTER #0 PARAMETERS FOR SCAN
00430 ; #####
00440 ;
3039 213640 00450 KBPFIX LD HL,4036H ; STORAGE FOR KEYSTROKE
303C 010139 00460 LD 9C,3901H ; FIRST ROW OF KEYS
303F 1600 00470 LD 0,0 ; COUNTER FOR COLUMNS
00480 ;
00490 ; #####
00500 ; CHECK EACH ROW OF KEYS IN SEARCH OF ONE THAT IS PRESSED
00510 ; #####
00520 ;
3041 0A 00530 KEYPRS LD A,(8C) ; RETRIEVE ROW CONTENTS
3042 5F 00540 LD E,A ; SAVE IT TEMPORARILY
3043 A3 00550 AND E ; SET FLAGS FOR TEST
3044 2D18 00560 JR NZ,STROKE ; NOT ZERO IF KEY PRESSED
3046 77 00570 LD (HL),A ; SAVE CURRENT VALUE
00580 ;
00590 ; #####
00600 ; INCREMENT AND ROTATE PATTERN CHECKS EACH ROW IN TURN
00610 ; #####
00620 ;
3047 14 00630 RECCHK INC D ; INCREMENT ROW COUNTER
3048 2C 00640 INC L ; INCREMENT STORAGE AREA
3049 C801 00650 RLC C ; GET NEXT KEYBOARD COLUMN
3048 78 00660 LD A,C ; GET VALUE INTO ACCUM.
00670 ;
00680 ; #####
00690 ; CHECK IF LAST VALID ROW (I.E., NOT INCLUDING SHIFT KEY)
00700 ; #####
00710 ;
304C D680 00720 SUB 80H ; LAST ROW IS 3980 HEX
304E 20F1 00730 JR NZ,KEYPRS ; NEXT CHECK IF NOT DONE
00740 ;
00750 ; #####
00760 ; AUTOREPEAT STATUS TEST ... CHECK IF KEYBOARD IS CLEAR
00770 ; #####
00780 ;
3050 0607 00790 LD B,7 ; COUNTER OF KBRO ROWS
3052 2D 00800 CLRMEM DEC L ; START COUNTING BACK
3053 96 00810 ADD A,(HL) ; AND ADD IT UP IN ACCUM
3054 10FC 00820 DJNZ CLRMEM ; AND DO IT FOR 7 ROWS
3056 A7 00830 AND A ; TEST FOR ANY KEY DOWN
3057 3E0D 00840 LD A,0 ; A=0, FLAGS ARE INTACT
3059 CD 00850 RET NZ ; BACK IF KEYS IN USE
00860 ;
00870 ; #####
00880 ; RESET AUTOREPEAT DELAY TO ZERO IF THE KEYBOARD IS CLEAR
00890 ; #####
00900 ;
305A 321A40 00910 LD (KPLACE),A ; ELSE DELAY GETS RESET
3060 C8 00920 RET ; AND GO BACK ANYWAY
00930 ;
00940 ; #####
00950 ; IF KEYSTROKE IS FOUND, CHECK STATUS OF AUTOREPEAT LOOP
00960 ; #####
00970 ;
306E A6 00980 STROKE AND (HL) ; CHECK KEYSTROKE STORAGE
305F 281E 00990 JR Z,FOUND ; NEW KEY IF NOT SAME
3061 3A8840 01000 LD A,(INKEYS) ; CHECK STATUS OF INKEY$
3064 A7 01010 AND A ; TEST IF SOMETHING THERE
3065 20E0 01020 JR NZ,RECCHK ; IF THERE IS, LOOP BACK
3067 3A1A40 01030 LD A,(KPLACE) ; NOW CHECK SPECIAL STORE
306A 3C 01040 INC A ; LET STORE = STORE + 1

```

Listing Continued . . .

It is in this distinction that the other patch can be made into the BASIC interpreter. All the DOS (disk operating system) commands *already exist* in Level II BASIC! A patch point (sometimes called a 'vector', other times a 'hook') is provided for each of these commands in RAM. When the disk system is added to the TRS-80, each of these patch points is filled with a jump to a DOS parameter.

Table 3-(?) presents a list of the DOS commands and their patch points in RAM. If you are not (and do not plan to be) a disk user, and if your programs will not be sold to potential disk users, then these patch points are for you. Each one can be used for your own set of commands, and every one will be accepted by a running BASIC program.

DOS COMMAND	List of DOS Patch Points	
	REPLACEMENT PATCH POINTS (HEX)	REPLACEMENT PATCH POINTS (DECIMAL)
CVI	4153 - 4154	16723 - 16724
FN	4156 - 4157	16726 - 16727
CVS	4159 - 415A	16729 - 16730
DEF	416C - 4150	16732 - 16733
CVD	415F - 4160	16735 - 16736
EDF	4162 - 4163	16738 - 16738
LOC	4165 - 4166	16741 - 16742
LOF	4168 - 4169	16744 - 16745
MKI\$	4168 - 416C	16747 - 16748
MKS\$	416E - 416F	16750 - 16751
MKO\$	4171 - 4172	16753 - 16754
CND	4174 - 4175	16756 - 16757
TIME\$	4177 - 4179	16759 - 16760
OPEN	417A - 4178	16762 - 16763
FIEL0	4170 - 417E	16765 - 16766
GET	4180 - 4191	16769 - 16768
PUT	4183 - 4184	16771 - 16772
CLOSE	4186 - 4187	16774 - 16775
LOAD	4189 - 419A	16777 - 16778
MERGE	418C - 4180	16780 - 16781
NAME	419F - 4190	16793 - 16794
KILL	4182 - 4193	16796 - 16797
&	4195 - 4199	16798 - 16799
LSET	4199 - 4199	16782 - 16793
RSET	4198 - 418C	16795 - 16786
INSTR	419E - 419F	16799 - 16799
SAVE	41A1 - 41A2	16801 - 16802
LINE	41A4 - 41A5	16804 - 16805

## Creating BASIC Tokens

Here's a program to start this discussion:

```

10 CLS : REM XXXXXXXXXXXXXXXX
20 Y = 15360
30 FOR X = 17129 TO 17300
40 POKE Y, PEEK (X)
50 Y = Y + 1 : NEXT X
60 PRINT "END OF PROGRAM"
70 PRINT "LIST OF PROGRAM"
80 LIST : REM ZZZZZZZZZZZZ

```

On the screen you now have two versions of the identical information - the BASIC program. The lines of X's and Z's are there to help you locate the program amidst some of what looks like garbage. You will also see that there are some familiar elements missing: the line numbers (which have been converted to hexadecimal), and all the BASIC commands

Continued Listing

```

306B 321A40 01050 LD [KPLACE],A ; AND PUT IT BACK THERE
306E FFFF 01060 CP OFFH ; CHECK IF IT IS AT END
3070 280B 01070 JR Z,DECA ; IF SO, THEN HOLD THERE
3072 C5 01080 PUSH BC ; SAVE ROW COUNTER REG.
3073 06FF 01090 LD B,OFFH ; GET DELAY VALUE INTO B
3075 10FE 01100 TMRWSTE DJNZ TMRWSTE ; AND DELAY JUST A BIT
3077 C1 01110 POP BC ; AND RESTORE ROW COUNTER
307B 18C0 01120 JR RCHEK ; AND BACK TO CHECK NEXT
307A 30 01130 DECA DEC A ; LET A = A - 1 (STORAGE)
307B 321A40 01140 LD [KPLACE],A ; AND PUT IT IN STORAGE
01150 ;
01160 ; #####
01170 ; GET KEYBOARD BYTE BACK, STORE AND PREPARE TO MANIPULATE
01180 ; #####
01190 ; FIRST CONVERT D FROM COUNTER TO PSEUDO ASCII EQUIVALENT
01200 ; #####
01210 ;
01220 LD A,E ; GET KEYBOARD BYTE BACK
01230 FOUNO LO [HL],E ; STORE IT IN STROKE AREA
01240 LD A,0 ; GET ROW COUNTER FROM D
01250 RLCA ; AND BEGIN A PROCESS...
01260 RLCA ; ...OF CONVERTING IT...
01270 RLCA ; ...TO AN OFFSET VALUE.
01280 LD 0,A ; AND PUT IT BACK IN D
01290 ;
01300 ; #####
01310 ; NOW PREPARE ROW COUNTER C TO COMPLETE ASCII CONVERSION
01320 ; #####
01330 ;
01340 LD C,1 ; GET NUMBER ONE READY
01350 BACKUP LO A,C ; ACCUM. HAS C FOR MATH
01360 AND E ; TEST IF C = KEYSTROKE
01370 JR NZ,AROUND ; IF NOT, THEN GO AROUND
01380 INC 0 ; ELSE D = ROW + COLUMN
01390 RLC C ; C SET TO NEXT COLUMN
01400 JR BACKUP ; GO BACK AND TEST AGAIN
01410 ;
01420 ; #####
01430 ; SHIFT ROW IS TESTED TO DETERMINE UPPER/LOWER STATUS
01440 ; #####
01450 ;
01460 AROUND LD A,(3B80H) ; GET SHIFT ROW FOR TEST
01470 LD B,A ; AND SAVE IT IN B
01480 LD A,0 ; GET ROW COUNTER BACK
01490 ADD A,40H ; AND CONVERT TO ASCII
01500 CP 60H ; IS IT UP/LW/GRAFIX/ETC
01510 JR NC,Z0429H ; GO OUT IF GRAPHICS MODE
01520 LD D,A ; SAVE PARTLY CONVERTED
01530 ;
01540 ; #####
01550 ; SHIFT/DOWN ARROW CHECKED FOR CONVERSION TO CONTROL CODE
01560 ; #####
01570 ;
01580 LD A,(3B40H) ; GET VALUE FOUND 7TH ROW
01590 AND 10H ; CHECK IF DOWN ARROW
01600 JR NZ,CNTR0L ; IF SO, PRODUCE CONTROL
01610 LD A,0 ; ELSE GET VALUE BACK
01620 RRC B ; B BUMPS INTO CARRY FLAG
01630 JR C,G0AWAY ; IF CARRY, THEN SHIFT
01640 ;
01650 ; #####
01660 ; LOWER CASE CONVERSION, MASK STRIPPING, FINAL TOUCHES
01670 ; #####
01680 ;
01690 ADD A,20H ; IF NOT THEN LOWER CASE
01700 JR G0AWAY ; AND GET OUT OF ROUTINE
01710 CNTR0L LO A,0 ; IF CONTROL CODE, GET IT
01720 SUB 40H ; GET RID OF ASCII MASK
01730 JR G0AWAY ; AND GET OUT OF ROUTINE
01740 Z0429H SUB 70H ; THE BALANCE OF THE
01750 JR NC,Z0430H ; ROUTINE BELOW UP TO
01760 ADD A,40H ; THE BEEP SECTION IS
01770 CP 3CH ; VIRTUALLY IDENTICAL
01780 JR C,Z0435H ; TO THE KEYBOARD
01790 XOR 10H ; DETERMINATION SUB-
01800 Z0435H RRC B ; ROUTINE FOUND IN-
01810 JR NC,G0AWAY ; ROM. A COMPLETE
01820 XOR 10H ; DESCRIPTION OF THIS
01830 JR G0AWAY ; SECTION OF THE KEY-
01840 Z0430H RLCA ; BOARD SCAN IS FOUND
01850 RRC B ; IN THE CHAPTER
01860 JR NC,Z0443H ; SUPPLEMENT ON THE
01870 INC A ; ROM KEYBOARD SCAN.
01880 LD HL,TABLET ; THIS TABLE IS CHANGED
01890 LD C,A ; FROM THE ONE FOUND
01900 LD B,0 ; IN EARLIER ROMS, BUT
01910 ADD HL,BC ; THE ROUTINE USED TO
01920 LD A,[HL] ; ACCESS IT IS THE
01930 JR G0AWAY ; SAME.
01940 ;
01950 ; #####
01960 ; TABLE BELOW DETERMINES TRS-80 (NOT ASCII) CONTROL CODES
01970 ; SEE SUPPLEMENT ON KEYBOARD SCAN FOR DETAILS ON CODES
01980 ; #####
01990 ;

```

(CLS, FOR, TO, POKE, PEEK, NEXT, PRINT, and LIST). What has happened to them?

For two purposes – economy of memory and speed of execution – legitimate BASIC commands are converted to single-byte keys called ‘tokens’. When you enter a BASIC command line, a subroutine evaluates each character group in that line, searching through all the keywords in ROM until it finds a match. When it finds a match, it replaces the original group of characters (PRINT, for example, which is five characters) with a single byte (178 in this case). Four bytes are saved, and the lengthy process of looking up the word PRINT is eliminated at run time.

Evaluating for tokens is indeed a time-consuming process. If you type 255 characters of garbage and press (ENTER), the computer will spend nearly two seconds attempting to tokenize that line before reporting a ?SN ERROR. A line which uses the command CHR\$( ) very often also takes time to tokenize. You can imagine the speed difference if this process were left to be done at RUN time.

The presence of tokens in a complicated program can make the difference between a running program and an ?OM ERROR. As an experiment, return to MEMORY SIZE? and respond with 17250. This gives you about 50 bytes of program space (at least 83 are needed to run any program). Enter these lines:

```

10 PRINT"THIS IS A TEST TO FIND OUT"
20 PRINT"HOW MUCH MEMORY SPACE IS IN HERE"

```

In attempting to run this program, you will get an ?OM ERROR. Now remove the word ‘IN’ and one space from the second line. The program will run fine. Finally, insert PRINT: on line 10. In spite of the fact that it *looks like* you have inserted 6 new characters (P-R-I-N-T-:), you have really only inserted two – the PRINT token (178) and a colon.

There is a lesson in this. If your program is running quite close to the end of your system’s available memory, try cutting down the lines of text within the program. Many more BASIC commands will then open up for use.

Another interesting trick opens up. You may have a BASIC program which you would like to convert for use on a printer. This can take up quite a bit of time. As a quick fix, you might just leaf through your program, replacing all PRINTs with LPRINTs. This won’t work every time (I’ll explain later), but it’s a useful

Listing Continued . . .

Continued Listing

```

3005 0000 02000 TABLET 0EWF 000DH ; CARR. RET. / CARR. RET.
3007 1F1F 02010 0EWF 1F1FH ; CLEAR SCRIN / CLEAR SCRIN
3009 0101 02020 0EWF 0101H ; BREAK KEY / BREAK KEY
300B 5B1B 02030 0EWF 1B5BH ; EDIT ESCAPE / UP ARROW
300D 0A00 02040 0EWF 000AH ; NOP (CHANGE) / LINEFEED
300F 0B1B 02050 0EWF 1B0BH ; BACKSP. LINE / BACKSP.
30E1 0919 02060 0EWF 1B0BH ; 32-CHAR MODE / HOR. TAB
30E3 2020 02070 0EWF 2020H ; SPACE / SPACE
02080 ;
02090 ; #####
02100 ; FINAL VALUE IS SAVED IN 0; STATUS OF SHIFT-0 TESTED
02110 ; #####
02120 ; #####
30E5 57 02130 GOAWAY LO 0,A ; SAVE VALUE IN 0 REG.
30E6 3A103B 02140 BEEEEP LO A,{3B10H} ; GET 0 KEYBOARD ROW
30E9 FED1 02150 CP 1 ; SEE IF IT IS ZERO (0)
30EB 2016 02160 JR NZ,BLEEPP ; GO OUT IF NOT ZERO
30ED 3A803B 02170 LD A,{3B80H} ; IF 0, CHECK SHIFT ROW
30F0 FED1 02180 CP 1 ; CHECK IF SHIFT KEY
30F2 200F 02190 JR NZ,BLEEPP ; IF NOT GO OUT TO BEEP
30F4 3A1940 02200 LD A,{SHIFTR} ; ELSE GET SHIFLOCK
30F7 EED1 02210 XOR 1 ; AND SWITCH 1-0 OR 0-1
30F9 321840 02220 LD {SHIFTR},A ; AND PUT IN SHIFLOCK
30FC 010005 02230 LD BC,500H ; GET LONGER DELAY
30FF CD6000 02240 CALL 0060H ; CALL ROM DELAY SUBR.
3102 C9 02250 RET ; AND GO BACK, NO BEEP
02260 ;
02270 ; #####
02280 ; DEBOUNCE IS ADDED; ALTERNATE: BEEP MAY BE LENGTHENED
02290 ; #####
02300 ; #####
3103 018001 02310 BLEEPP LD BC,180H ; DEBOUNCE VALUE AND
3106 CD6000 02320 CALL 0060H ; DELAY CALL TO ROM
3109 7A 02330 LD A,0 ; GET STORED VALUE BACK
310A C5 02340 PUSH BC ; SAVE BC REGISTER
310B F5 02350 PUSH AF ; SAVE ACCUM AND FLAGS
310C 0640 02360 LO B,40H ; GET NOTE LENGTH VALUE
310E 3A3D40 02370 LO A,{PORTFF} ; GET STATUS OF SCREEN
3111 E6F0 02380 AND 0F0H ; MASK SCREEN CHANGE OUT
3113 67 02390 LD H,A ; STORE MSB IN H REG.
3114 F602 02400 OR 2 ; SET BIT 1 TO BE ON
3116 6F 02410 LD L,A ; STORE ALT. MSB IN L REG
3117 70 02420 BEEPER LD A,L ; GET ALT. MSB TO OUTPUT
3118 03FF 02430 OUT {OFFH},A ; AND OUTPUT RISING WAVE
311A 7C 02440 LD A,H ; NOW GET NORMAL MSB
311B 03FF 02450 OUT {OFFH},A ; AND OUTPUT FALLING WAVE
3110 C5 02460 PUSH BC ; SAVE NOTE LENGTH REG.
311E 0640 02470 LO B,40H ; GET FREQUENCY DELAY
3120 10F2 02480 FREQCY DJNZ FREQCY ; AND WAIT A LITTLE WHILE
3122 C1 02490 POP BC ; NOW RESTORE LENGTH VAL.
3123 10F2 02500 DJNZ BEEPER ; AND GO BACK THAT LENGTH
3125 F1 02510 POP AF ; RESTORE ORIGINAL CHAR.
3126 C1 02520 POP BC ; AND RESTORE ORIGINAL BC
3127 C35204 02530 JP 0452H ; BACK TO ROM IN PROGRESS
02540 ;
02550 ; #####
02560 ; THIS IS LOWER CASE DETERMINATION FROM STORED INFO
02570 ; #####
02580 ; #####
312A F5 02590 LOWER PUSH AF ; SAVE NEEDED REGISTER
312B 3A1940 02600 LD A,{SHIFTR} ; GET STATUS OF SHIFLOCK
312E FED1 02610 CP 1 ; CHECK IF STATUS = 1
3130 2B04 02620 JR Z,LOWER1 ; IF SO THEN GO TO L.C.
3132 F1 02630 POP AF ; ELSE GET ORIGINAL VALUE
3133 C35804 02640 JP 045BH ; LEAVE TO NORMAL DISPLAY
3136 F1 02650 LOWER1 POP AF ; ELSE GET ORIGINAL VALUE
3137 0D6E03 02660 LD L,{IX+3} ; GET CURSOR LSB INTO L
313A 0D6E04 02670 LD H,{IX+4} ; GET CURSOR MSB INTO H
313D 0A8A04 02680 JP C,049AH ; BACK TO ROM IF CARRY
3140 007E05 02690 LD A,{IX+5} ; GET CURSOR CHARACTER
3143 B7 02700 OR A ; TEST IF CURSOR IS ON
3144 2B01 02710 JR Z,GETCHR ; IF NOT THEN GO 00 IT
3146 77 02720 LD {HL},A ; ELSE PUT IT BACK
3147 79 02730 GETCHR LD A,C ; GET VALUE TO DISPLAY
3148 FE20 02740 CP 20H ; IS IT A SPACE OR CNTRL?
314A 0A0605 02750 JP C,0506H ; OUT IF SPACE OR CONTROL
3140 FE80 02760 CP 80H ; IS IT GRAPHICS OR TAB?
314F 02A604 02770 JP NC,04A6H ; OUT IF GRAPHICS OR TAB
3152 C37004 02780 JP 0470H ; DO UNCONVERTED DISPLAY
02790 ;
02800 ; #####
06CC 02810 ENO 06CCH ; BACK TO BASIC READY
00000 TOTAL ERRORS
24295 TEXT AREA BYTES LEFT
AROUND 3090 01460 01370
BACKUP 3087 01350 01400
BEEEEP 30E5 02140
BEEPER 3117 02420 02500
BLEEEP 3103 02310 02160 02190
CLRMEM 3052 00800 00820
CNTRL 30AC 01710 01600
OECA 307A 01130 01070
FOUNO 307F 01230 00980
FREQCY 3120 02480 02480
GETCHR 3147 02730 02710

```

crude first pass; enter this line from command level (no line number):

```
FOR X=17130 TO 32767 : IF PEEK(X)=17B THEN POKE X,175 : NEXT X
ELSE NEXT
```

This line will search through all of memory (in a 16K machine), looking for the PRINT token (via PEEK), and replacing it wherever it finds it with the LPRINT token (via POKE).

There are occasional risks with this kind of POKEing. Occasionally, a line number or part of the BASIC line organization addresses may have the same value as the PRINT token and get changed with this process. You might end up with a line like 37549 in the middle of a nicely ordered sequence of 1000, 1010, 1020, etc. Another possible flaw is that the value stored as a pointer to the next BASIC line may also correspond to the PRINT token, and get changed. You may get into a LIST-loop or 'lose' some lines (at least to the eye - they are still in there).

If a line number is wrong, merely delete the incorrect line number and retype a new line with the correct number. If some lines seem lost or the LIST command keeps looping, then temporarily reverse the process . . .

```
FOR X=17130 TO 32767 : IF PEEK(X)=175 THEN POKE X,17B : NEXT X
```

. . . and add a few REM statements in a line before the error occurred. Adding statements will alter the position of program lines in memory, and you will probably be able to perform the original conversion again safely.

As you can see, the tokenizing process has a lot of distinct advantages. As a postscript, consider the program that opened this section. The tokens were displayed as graphics characters. Why is this so?

Since there are only 256 possible combinations of bits in a byte, many times they have to serve multiple masters. In a machine language program, these bytes can be instructions. In a BASIC program, they are tokenized commands. On screen - which means in video memory - they appear as graphics. The 26 letters of our alphabet can be combined and recombined to form words, sentences, paragraphs, etc., and many words can sound alike or be spelled alike and still have different meanings. Context changes how words are understood.

Douglas Hofstadter played on this most dramatically when he wrote, "This sentence no verb".

Listing Continued . . .

## Continued Listing

```

GOAWAY 30E5 D213D D1630 017DD D173D 01810 0183D 0183D
INKEYS 40B9 00250 01000
KBPFIX 3038 00450 DD35D
KEYPRS 3D41 DD53D DD73D
KPLACE 401A DD27D DD81D D1030 D1050 D1140
LDWER 312A D2590 DD370
LOWER1 3136 02650 02620
PORTFF 4030 DD26D D2370
RECEK 3047 00630 01020 D1120
SHIFTR 4018 DD28D 0220D 02220 D26DD
STROKE 305E DD88D DD56D
TABLET 3005 D200D D188D
TMSWE 3075 D110D 01100
Z0429H 3DB1 D1740 01510
Z0435H 3DB0 0180D 0178D
Z043DH 30C5 0184D 0175D
Z0443H 30C8 0188D 0186D

001DD ; #####
00110 ; ADDING DEBOUNCE/BEEP/AUTOREPEAT TO EDITOR/ASSEMBLER 1.1
00120 ; #####
00130 ; NEW EDTASM SCAN PATCH WILL USE ROM ROUTINE IN ITS PLACE
00140 ; #####
00150 ;
4016 0016D ORG 4016H ; KEYBOARD PATCH POINT
4016 68 0017D DEF8 068H ; LSB OF NEW START
4017 50 0018D DEF8 050H ; MSB OF NEW START
5D68 0019D ORG 5068H ; NEW PARTIAL K80 DRIVER
6068 211640 0020D LD HL,4016H ; FORMER K80 PATCH POINT
5D68 36E3 0021D LD (HL),DE3H ; LSB OF ADDRESS IN ROM
5D60 23 0022D INC HL ; GET NEXT POSITION READY
506E 3603 0023D LD (HL),03H ; MSB OF ADDRESS IN ROM
5D7D C38A46 0024D JP 468AH ; JUMP TO EDTASM PROGRAM
00250 ;
00260 ; #####
0027D ; NEW KEYBOARD SCAN IS PLACED AT FORMER SOURCE CODE START
0028D ; #####
0029D ;
500D 0030D ORG 5000H ; NEW BLOCK OF CODE
500D 212140 0031D LD HL,4021H ; GET REPEAT STORAGE BYTE
5003 D1D138 0032D LD BC,3B01H ; GET FIRST ROW OF KEYBRD
5006 16DD 0033D LO D,D ; ROW COUNTER REGISTER
5D08 0A 0034D LD A,(BC) ; FIND KEY PRESSED IN ROW
5D08 5F 0035D LD E,A ; SAVE VALUE IN E REG.
5D0A A3 0036D AND E ; TEST FOR PRESSED KEY
5D08 201B 0037D JR NZ,FOUND ; JUMP OUT IF KEY PRESSED
500D 77 0038D LD (HL),A ; SAVE CURRENT KEY IN HL
50DE 14 0039D REDOIT INC 0 ; INCREMENT ROW COUNTER
5DDF 2C 0040D INC L ; INCREMENT KEY STORAGE
5D10 C8D1 0041D RLC C ; GET NEXT ROW INTO BC
5D12 78 0042D LD A,C ; GET VALUE OF C FOR TEST
5D13 068D 0043D SUB BDH ; BDH IS SHIFT KEY ROW
5015 20F1 0044D JR NZ,LDOFX ; LOOP BACK IF NOT SHIFT
5017 06D7 0045D LD B,7 ; NUMBER OF BUMPS TO DD
5D18 20 0046D LDOFY DEC L ; DECREMENT BYTE STORAGE
5D1A B6 0047D ADD A,(HL) ; ADD TOTAL VALUE STORED
5018 10FC 0048D DJNZ LDOFY ; LOOP BACK FOR 7 TIMES
5D1D FEDD 0049D CP D ; WERE NO KEYS PRESSED?
501F 3EDD 0050D LD A,0 ; CLEAR ACC., NOT FLAGS
5D21 C0 0051D RET NZ ; BACK TO MAIN ROUTINE
5D22 321A40 0052D LD (401AH),A ; SAVE VALUE IF ZERO
5D25 C9 0053D RET ; BACK TO MAIN ROUTINE
5D26 A6 0054D FOUND AND (HL) ; TEST IF SAME CHARACTER
5D27 2810 0055D JR Z,SAME ; IF SAME, JUMP OUT ...
5D29 3A1A40 0056D LO A,(401AH) ; GET COUNTER BYTE INTO A
5D2C 3C 0057D INC A ; INCREMENT IT EACH TIME
5D2D 321A40 0058D LD (401AH),A ; SAVE IT AGAIN FOR NEXT
5D30 FEFF 0059D CP OFFH ; IS IT PAST FULL DELAY?
5032 20DA 0060D JR NZ,REDOIT ; IF NOT GO BACK FOR MORE
5034 30 0061D DEC A ; DEC A TO DFE FOR REPEAT
5D35 321A40 0062D LD (401AH),A ; SAVE THAT VALUE IN CNTR
5D38 78 0063D LD A,E ; GET ORIGINAL CHARACTER
5039 73 0064D SAME LO (HL),E ; SAVE THAT VALUE IN HL
603A C5 0065D PUSH BC ; SAVE ROW COUNTER
503B D1DD2 0066D LD BC,D2DDH ; GET DELAY VALUE READY
503E CD6000 0067D CALL 006DH ; CALL ROM DELAY SUBROUT.
5D41 C1 0068D POP BC ; RESTORE ROW COUNTER
5D42 DA 0069D LD A,(BC) ; GET VALUE AT ROW CNTR.
5043 A3 007DD AND E ; TEST IF STILL PRESSED
6044 C8 0071D RET Z ; IF NOT THEN IT'S BOUNCE
6D45 C5 0072D PUSH BC ; SAVE ROW COUNTER AGAIN
5D48 E5 0073D PUSH HL ; SAVE STORAGE AREA
5D47 F6 0074D PUSH AF ; SAVE CURRENT KEYSTRDKE
6048 D640 0075D LD B,40H ; GET DURATION VALUE
5D4A 3A3D40 0076D LD A,(403DH) ; GET STATUS OF 6CSCREEN
504D E6FD 0077D AND 0FDH ; CLEAR THE DATA OUTPUT
6D4F 67 0078D LD H,A ; H BECOMES OUTPUT MASK
5D50 F602 0079D DR 02 ; READY ACC. FOR BIT-6ET

```

Listing Continued...

## Packing BASIC With Machine Code

One of the most attractive aspects of interpreted code like Level II BASIC is the hide-n-seek game you can play with it. One of the most fruitful games is called 'string packing', a technique that allows machine language programs to be hidden inside ordinary program lines.

It is convenient and efficient, but once it's part of a program, it looks very obscure. There are three ways of creating machine-coded programs through BASIC strings, but all depend on the code being relocatable (see Supplement to Chapter 4 regarding relocatable code). The three ways are:

1. Packing the machine code into a string on a program line, one character at a time. This is done when the program is created.
2. Packing the machine code into a string on a program line, one character at a time, reading data on other program lines. This is done each time the program is run. The read/data lines containing the packing information are then automatically deleted.
3. Building a string in the string variable area from in-line CHR\$( ) commands. The strings are built each time the program is run.

First, some background on strings and how to find them. In the TRS-80, all variables can be located with a Level II command known as VARPTR (VARiable PoinTeR). The variable pointer can find out a lot about variables – their type, location, and length. In the case of string variables, VARPTR returns a memory location in which the length of the string is stored.

Assume A\$ is a string variable in a program line. The statement X=VARPTR(A\$) assigns the address of the first part of the A\$ story to X. Here's what X can reveal:

```

PEEK (X)    ...the length of A$
PEEK (X+1)  ...the least significant byte of where A$ is found
PEEK (X+2)  ...the most significant byte of where A$ is found

```

These values are returned in decimal form, because Level II doesn't provide a hexadecimal numbering option. To find where A\$ is actually located, then, use this formula:

$$AD = \text{PEEK}(X+1) + 256 * \text{PEEK}(X+2)$$

Do you see what is happening here? And what can be done with it? If you know, for example, that A\$="XXXXXXXXXX", then you can actually *change* A\$ by POKEing values into the

## Continued Listing

```

5052 6F 00800 LD L,A ; L BECOMES OUTPUT MASK
5053 70 00810 SOUND LD A,L ; GET BIT-SET MASK
5054 03FF 00820 OUT (0FFH),A ; OUTPUT IT (WAVEFORM HI)
5056 7C 00830 LD A,H ; GET BIT-RESET MASK
5057 03FF 00840 OUT (0FFH),A ; OUTPUT IT (WAVEFORM LO)
5059 C5 00850 PUSH BC ; SAVE DURATION VALUE
505A 0640 00860 LD B,40H ; GET PITCH VALUE
505C 10FE 00870 DJNZ $ ; DELAY FOR AUDIBLE TONE
505E C1 00880 POP BC ; RESTORE BEEP DURATION
505F 10F2 00890 DJNZ SOUND ; LDOP FOR FULL DURATION
5061 F1 00900 POP AF ; RESTORE KEYSTROKE VALUE
5062 E1 00910 POP HL ; RESTORE STORAGE VALUE
5063 C1 00920 POP BC ; RESTORE ROW COUNTER
5064 C3D744 00930 JP 4407H ; JUMP INTO EDTASM...!
00940 ;
00950 ; #####
00960 ; FOLLOWING PATCH NEW SOURCE CODE START & KEYBOARD SCAN
00970 ; #####
00980 ;
468A 00990 ORG 468AH ; PLACE TO PATCH END
468A 21005E 01000 LD HL,5E00H ; NEW END OF EDTASM
4710 01010 ORG 4710H ; PLACE TO PATCH END
4710 11005E 01020 LD DE,5E00H ; NEW END OF EDTASM
4A07 01030 ORG 4A07H ; PLACE TO PATCH END
4A07 11005E 01040 LD DE,5E00H ; NEW END OF EDTASM
4A0B 01050 ORG 4A0BH ; PLACE TO PATCH END
4A0B 21005E 01060 LD HL,5E00H ; NEW END OF EDTASM
4B50 01070 ORG 4B50H ; PLACE TO PATCH END
4B50 21005E 01080 LD HL,5E00H ; NEW END OF EDTASM
4C39 01090 ORG 4C39H ; PLACE TO PATCH END
4C39 21005E 01100 LD HL,5E00H ; NEW END OF EDTASM
4D80 01110 ORG 4D80H ; PLACE TO PATCH END
4D80 21005E 01120 LD HL,5E00H ; NEW END OF EDTASM
5227 01130 ORG 5227H ; PLACE TO PATCH END
5227 21005E 01140 LD HL,5E00H ; NEW END OF EDTASM
43EF 01150 ORG 43EFH ; PLACE TO PATCH KBD SCAN
43EF C3D050 01160 JP 5000H ; NEW KEYBOARD SCAN
01170 ;
01180 ; #####
5000 01190 END 5000H
00000 TOTAL ERRORS
30935 TEXT AREA BYTES LEFT

FOUND 5026 00540 00370
LOOPX 5008 00340 00440
LOOPY 5019 00460 00480
REEDIT 50DE 00390 006D0
SAME 5039 00640 00550
SOUND 5053 00810 00890

```

address you have calculated.

Run the following demonstration program:

```

10 CLS
20 A$ = "XXXXXXXXXX"
30 X = VARPTR(A$)
40 Q = PEEK (X)
50 AD = PEEK (X+1) + 256 * PEEK (X+2)
60 PRINTA$
70 L = 65
80 FOR N = 1 TO Q
90 POKE AD,L
100 L = L + 1
110 AD = AD + 1
120 PRINT A$
130 NEXT N
140 LIST 20

```

A\$ is created in line 20, and its information storage area is found by X in line 30. Its length is discovered in line 30 and assigned to variable Q; its location is determined in line 50 (assigned to variable AD), and it is printed in its original form in line 60. Line 70 assigns the value 65 (an ASCII 'A') to variable L, and a Q-character loop is created in line 80. The first character in A\$ is POKed with L, i.e., changed to letter 'A'. L is incremented (to letter 'B'), AD is incremented (to the next character in A\$), and the new A\$ is printed. When all characters in A\$ have been changed, it is listed. You can see that the list itself is changed because A\$ is defined and stored right there in line 20!

As a further experiment, change the value of L in line 70 to 129 and watch what happens (see Box on Tokenizing for details).

Here is the point: you can create a dummy string like A\$ which will become the residence of a machine language program. By finding A\$ and POKEing your program into it, it can be CLOAded and CSAVED at will.

Now we turn to the three string-packing methods themselves. For the sample program, the following 20-byte routine will be used:

```

21 D1 3C      LD      HL,3CD1H
E5           PUSH    HL
01           PDP     DE
2B           DEC     HL
01 FF 03     LD      BC,03FFH
36 20        LD      (HL),20H
E0 80        LDIR
21 11 D1     LD      HL,D111H
C0 A7 2B     CALL    2BA7H
C9           RET

```

This routine clears the screen, prepares the message 'RADIO SHACK LEVEL II BASIC', and displays it.

The first and second string-packing methods are essentially identical, except that in the first method, A\$ is created and the program saved for later use. The second method creates A\$ at every program run. Here is a simple BASIC program using the routine above packed into A\$:

## Packing BASIC With Machine Code

```

00100 ; #####/
00110 ; ELEMENTARY (AND SLOW) FOUR-VOICE MUSIC SUBROUTINE.
00120 ; THIS ROUTINE IS CURRENTLY SET UP FOR FOUR INDEPENDENT
00130 ; VOICES. BECAUSE OF TIMING CONSTRAINTS, VOICES ARE VERY
00140 ; LOW IN PITCH. BY USING THE FOUR SUBROUTINES ALTERNATE-
00150 ; LY FOR SOUND EFFECTS, A HIGHER SPEED CAN BE OBTAINED.
00160 ; A TEST-AND-JUMP LOOP CAN BE INSERTED FOR THIS PURPOSE.
00170 ; ALSO, AS LONG AS LABELS ARE CHANGED, VOICES CAN BE
00180 ; DROPPED FROM THE SEQUENCE, RESULTING A 50%, 100% AND
00190 ; 150% SPEED INCREASE, RESPECTIVELY. SQUARE WAVE OUTPUT.
00200 ; NOTICE ALSO THAT, WHEN VOICES ARE DROPPED, OTHER REG-
00210 ; ISTERS MAY BE SUBSTITUTED FOR IX, FOR A SPEED INCREASE.
00220 ; FURTHERMORE, CHANGING THE SPEED OF THE PROCESSOR WILL
00230 ; PROVIDE AN IDENTICAL INCREASE IN THE OUTPUT PITCH.
00240 ; #####/
00250 ;
00260 ORG 4F60H ; :NEAR TOP OF MEMORY
00270 LO IX,5000H ; :START PITCH & RHYTHM
00280 LO 8C,4FFFH ; :MEMORY-MAPPED SOUND
00290 ;
00300 ; #####/
00310 ; OUTER (INTER-NOTE) LOOP BEGINS HERE; T-STATES 212 - 244
00320 ; #####/
00330 ;
00340 LOOP1 EXX ;04:READY DURATION REGS.
00350 LO 8,(IX+0) ;19:MSB OF NOTE DURATION
00360 LO C,(IX+1) ;19:LSB OF NOTE DURATION
00370 EXX ;04:STASH REGISTER AWAY
00380 LO H,(IX+2) ;18:FIRST PITCH INTO H
00390 LO L,(IX+3) ;19:SECONDO PITCH INTO L
00400 LO O,(IX+4) ;19:THIRO PITCH INTO O
00410 LO E,(IX+5) ;18:FOURTH PITCH INTO E
00420 ;
00430 ; #####/
00440 ; EACH VALUE ACQUIRED FROM IX IS TESTED TO SEE IF IT IS 0
00450 ; AND THE VOICE IS TURNED OFF IF IT IS (DEFINING A REST).
00460 ; #####/
00470 ;
00480 LO A,(BC) ;07:READY TO TWEAK MEM
00490 ANO 0FH ;04:TURN ALL VOICES ON
00500 INC H ;04:BUMP VALUE; REST TEST
00510 DEC H ;04:BUMP VALUE; REST TEST
00520 JP HZ,REST1 ;10:ONLY 00 DEFINES REST
00530 SET 4,A ;08:SILENCE VOICE IF REST
00540 REST1 INC L ;04:BUMP VALUE; REST TEST
00550 DEC L ;04:BUMP VALUE; REST TEST
00560 JP NZ,REST2 ;10:ONLY 00 DEFINES REST
00570 SET 5,A ;08:SILENCE VOICE IF REST
00580 REST2 INC O ;04:BUMP VALUE; REST TEST
00590 DEC O ;04:BUMP VALUE; REST TEST
00600 JP NZ,REST3 ;10:ONLY 00 DEFINES REST
00610 SET 6,A ;08:SILENCE VOICE IF REST
00620 REST3 INC E ;04:BUMP VALUE; REST TEST
00630 DEC E ;04:BUMP VALUE; REST TEST
00640 JP NZ,REST4 ;10:ONLY 00 DEFINES REST
00650 SET 7,A ;08:SILENCE VOICE IF REST
00660 REST4 LO (8C),A ;07:SET VOICES ON OR OFF
00670 ;
00680 ; #####/
00690 ; DECREMENT H,L,O,E (WAVEFORM DURATION FOR EACH VOICE)...
00700 ; NEEDED EACH TIME THE WAVEFORM IS TOGGLED DURING LOOPS...
00710 ; ...INNER LOOP BEGINS HERE. T-STATES STRICTLY EQUAL 246
00720 ; MEANING MAXIMUM FREQUENCY IS APPROXIMATELY 1770000/246
00730 ; OR 7195.1 HZ. USEFUL FREQUENCIES ARE CONSIDERABLY LESS.
00740 ; #####/
00750 ; BEGIN PITCH AND RHYTHM COUNTDOWN LOOPS
00760 ; #####/
00770 ; COUNT DOWN THE PITCH LOOP FOR VOICE NUMBER ONE
00780 ; #####/
00790 ;
00800 LOOP2 LO A,(BC) ;07:WHAT WAVE IS LURKING
00810 DEC H ;04:COUNTDOWN FREQUENCY 1
00820 JP HZ,EXIT1 ;10:SAME WAVE IF NOT 0
00830 XOR 1 ;07:TOGGLE WAVEFORM BIT 1
00840 LO H,(IX+2) ;19:RESTORE PITCH VALUE
00850 JP EXIT1A ;10:JUMP PAST TIMEWASTERS
00860 EXIT1 PUSH IY ;15:WASTE 15 T-STATES
00870 POP IY ;14:WASTE 14 T-STATES
00880 ANO 0FFH ;07:WASTE 7 MORE T-STATES
00890 ;
00900 ; #####/
00910 ; COUNT DOWN THE PITCH LOOP FOR VOICE NUMBER TWO
00920 ; #####/
00930 ;
00940 EXIT1A DEC L ;04:COUNTDOWN FREQUENCY 2
00950 JP NZ,EXIT2 ;10:SAME WAVE IF NOT 0
00960 XOR 2 ;07:TOGGLE WAVEFORM BIT 2
00970 LO L,(IX+3) ;19:RESTORE PITCH VALUE
00980 JP EXIT2A ;10:JUMP PAST TIMEWASTERS
00990 EXIT2 PUSH IY ;15:WASTE 15 BANANAS
01000 POP IY ;14:DRUM FINGERS ON 14
01010 ANO 0FFH ;07:USELESS ARITHMETIC
01020 ;
01030 ; #####/
01040 ; COUNT DOWN THE PITCH LOOP FOR VOICE NUMBER THREE
01050 ; #####/
01060 ;
01070 EXIT2A DEC O ;04:COUNTDOWN FREQUENCY 3
01080 JP NZ,EXIT3 ;10:SAME WAVE IF NOT 0
01090 XOR 4 ;07:TOGGLE WAVEFORM BIT 3
01100 LO O,(IX+4) ;19:RESTORE PITCH VALUE
01110 JP EXIT3A ;10:JUMP PAST TIMEWASTERS
01120 EXIT3 PUSH IY ;15:SCRATCH LEFT HAND
01130 POP IY ;14:SCRATCH RIGHT HAND

```

```

5 POKE 16553,255 : REM OPTIONAL LINE
10 PRINT"THE PROGRAM IS RUNNING"
20 A$ = "12345678901234567890"
30 X = VARPTR (A$) : Q = PEEK (X)
40 AO = PEEK (X+1) + 256 * PEEK (X+2)
50 FOR N = 1 TO Q : READ A
60 POKE AO,A : AO = AO + 1 : NEXT
70 HB = PEEK (X+1) : POKE 16526,HB
80 LB = PEEK (X+2) : POKE 16527,LB
90 INPUT"ENTER TO RUN M/L";Z
100 M$ =USR (0)
110 DATA 33,1,80,229,209,43,1,255,3,54
120 DATA 32,237,178,33,17,1,205,167,40,201

```

Line 5 is optional if you have an early ROM set, where data reads could RESTORE after every READ. Line 20 contains the string to be packed, and, as before, lines 30 and 40 identify the string's length and location. The data is read and POKEd into place sequentially by lines 50 and 60.

Finally, lines 70 and 80 identify the beginning of the string and place it in the USR(X) entry points at 16526 and 16527. The program pauses for user input in line 90, and then jumps to the packed routine.

After the program has been run, list it. Note that A\$ is now packed with new information replacing the string '12345678901234567890'. The first string-packing method saves space by deleting lines 40 through 60, 110 and 120, which have done their work. The program is then CSAVED, and can be loaded and run at any time. The second string-packing method leaves all lines intact so that any future users may modify them as necessary.

There are a few disadvantages to this method of string packing. First of all, two machine language instructions or pieces of data may not be used directly: 00 and 22. 00 tells a BASIC program it has found the end of a program line; two 00's in a row indicate the end of the program. 22 is the quotation mark symbol, and will inform the program that the string has ended; a ?SN ERROR will then be produced in the rest of the line.

A second difficulty is that the line containing A\$ may not be edited. This is because when a line is edited, it is placed from the LIST into a buffer that acts exactly like the keyboard buffer; the bytes within the quotes are then converted into the individual letters. For example, code (178) is a machine command which also is the BASIC token for PRINT; when listed, it comes up on the screen as PRINT. Editing the line puts P-R-I-N-T in the edit buffer; but since it is *within the quotation marks*, it is not tokenized. The result? The string now contains five ASCII characters where it once contained a machine language instruction!

Listing Continued . . .

## Continued Listing

```

4F00 E6FF      01140      AND      OFFH      ;07:CHECK KITCHEN CLOCK
                01150 ;
                01160 ; #####
                01170 ; COUNT DOWN THE PITCH LOOP FOR VOICE NUMBER FOUR
                01180 ; #####
                01190 ;
4F02 10        01200 EXIT3A DEC      E      ;04:COUNTDOWN FREQUENCY 4
4F03 C20E4F    01210      JP      NZ,EXIT4    ;10:SAME WAVE IF NOT 0
4F06 EE08      01220      XOR      B      ;07:TOGGLE WAVEFORM BIT 4
4F08 005E05    01230      LD      E,(IX+5)    ;19:RESTORE PITCH VALUE
4F08 C3E44F    01240      JP      EXIT4A      ;10:JUMP PAST TIMEWASTERS
4F0E F0E5      01250 EXIT4  PUSH     IY      ;15:WATER NASTURTIUMS
4FE0 F0E1      01260      POP      IY      ;14:PICK 14 ZUCCHINI
4FE2 E6FF      01270      AND      OFFH      ;07:MIX APPLES AND ORANGE
                01280 ;
                01290 ; #####
                01300 ; CHECK FOR END OF NOTE DURATION; GET MORE NOTES IF 00NE
                01310 ; #####
                01320 ;
4FE4 02        01330 EXIT4A  LD      (BC),A    ;07:OUTPUT NEW WAVEFORMS
4FE5 09        01340      EXX             ;04:GET STASHE DURATION
4FE6 08        01350      DEC      BC      ;06:COUNT DOWN DURATION
4FE7 78        01360      LD      A,B      ;04:SET UP B FOR TEST
4FE8 81        01370      OR      C      ;04:CHECK AGAINST C
4FE9 09        01380      EXX             ;04:STASH DURATION AGAIN
4FEA C29B4F    01390      JP      NZ,LOOP2    ;10:GO BACK TIL NOTE END
                01400 ;
                01410 ; #####
                01420 ; MOVE ALL POINTERS PAST CURRENT BATCH OF NOTES/DURATIONS
                01430 ; ...THIS IS THE REMAINDER OF OUTER LOOP, T-STATES = 80,
                01440 ; TOTAL T-STATES OF OUTER LOOP = 80 + 244 = 324, WHICH IS
                01450 ; ABOUT 2 OF THE MAXIMUM CYCLE FREQUENCIES (.0002 USEC).
                01460 ; #####
                01470 ;
4FE0 110600    01480      LD      DE,6      ;10:MEMORY POS'NS TO MOVE
4FF0 0019      01490      ADD      IX,DE     ;15:MOVE 6 PLACES FORWARD
                01500 ;
                01510 ; #####
                01520 ; CHECK FOR END OF PROGRAM CODE (00) OR DEPRESSE BREAK
                01530 ; #####
                01540 ;
4FF2 007E00    01550      LD      A,(IX+0)    ;19:NEXT NOTE DURATION
4FF5 87        01560      OR      A      ;04:SET END-OF-MUSIC FLAG
4FF6 C8        01570      RET      Z      ;05:BACK TO BASIC IF 00NE
4FF7 3A403B    01580      LD      A,(3B40H)   ;13:TEST BREAK KY80 ROW
4FFA 87        01590      OR      A      ;04:SET FLAG FOR KEY TEST
4FFB CA674F    01600      JP      Z,LOOP1    ;10:CONTINUE PIECE IF OK
4FFE C9        01610      RET             ; :TO BASIC IF BREAK
                01620 ;
                01630 ; #####
                01640 ;
06CC          01650      END      06CCH      ; :READY AFTER SLASH
00000 TOTAL ERRORS

```

```

30 X = VARPTR (A$)
40 POKE 16526, PEEK (X+1)
50 POKE 16527, PEEK (X+2)
60 INPUT "ENTER TO RUN M/L";Z
70 M$ =USR(0)

```

This time, A\$ has been defined in the BASIC variable area using CHR\$(). In other words, because A\$ is no longer defined within the program line, *any value may be used in the machine language program*. Hybrid strings can also be used, as:

```
A$ = "1234567890" + CHR$(0) + "12345" + CHR$(34)
```

This method uses both the POKEing via READ-DATA statments, plus concatenation with 00 and 22 where necessary. Once again, A\$ is stored in the variable area, and none of the 00's or 22's affect the BASIC program. As a final amusement, RUN listing 3-(?) above, then PRINT A\$. The A\$ in listing 3-(?) and the A\$ in this listing both contain identical machine code. RUN listing 3-(?) again. LIST line 20. Now PRINT A\$. The answer is up to you.

String-packing will likely become a very important addition to your library of BASIC tools. Here, then, is a summary of the string-packing technique:

1. Write the BASIC program.
2. Create a dummy string of any unused variable name (for example, A\$="LOONIETUNES")
3. Make the string the exact length of your machine language program.
4. Write a program line that sets another variable to point to the string's variable information.  
For example, X = VARPTR(A\$).
5. Find the starting address of the string by converting the decimal bytes into a single decimal value.  
AD = PEEK(X+1) + 256 \* PEEK (X+2).
6. Create a set of READ and DATA lines in your BASIC program which will POKE the machine language program into place in the dummy string. (For example:

```

Line #   FOR N = AD TO AD+3 : READ L : POKE N,L : NEXT
Line #   DATA 33, 16, 16, 204 .....

```

7. Set the USR(0) entry point at 16526 and 16527 to the beginning of the string variable storage address (for example, PEEK (X+1) and PEEK (X+2), respectively.
8. If you wish, delete the READ, DATA and POKE loop lines used for that routine.  
CSAVE the program.

The last method of string packing is capable of overcoming both these flaws. Examine the listing below:

```

10 PRINT"THE PROGRAM IS RUNNING"
20 A$ = CHR$(33) + CHR$(1) + CHR$(60) +
      CHR$(229) + CHR$(209) + CHR$(43) +
      CHR$(1) + CHR$(255) + CHR$(3) +
      CHR$(54) + CHR$(32) + CHR$(237) +
      CHR$(176) + CHR$(33) + CHR$(17) +
      CHR$(1) + CHR$(205) + CHR$(167) +
      CHR$(40) + CHR$(201)

```

## Sound-Effects Generation

The essence of sound generation has already been sneaked in: the audible beep in the keyboard routine at the beginning of this Chapter. Sound has been something of a mystery, but there could hardly been a simpler machine language program. Try this experimental program with the tape recorder running in record mode (the motor plug removed), and a cassette in place:

```
10 FORX=1TO2000:OUT255,0:OUT255,255:NEXT
```

In order that this program can operate at top speed, make sure you do not use spaces, and keep the program on one line. You'll hear the tape relay clatter and see the screen jitter. Now play back the short segment of tape you recorded. There is a buzzing on that tape, similar in pitch to the buzzing of the cassette relay. What does all this mean?

It is quite simple. There is an electronic switch inside the TRS-80 which controls several activities: whether the screen is in 64 or 32 character mode, whether the cassette relay is on or off, and whether the cassette data output is 'on' or 'off'.

I've put on and off in quotation marks for a reason: which video mode is used and whether the cassette motor is on or off are examples of states or conditions. But in audio terms, the latter output represents very swift sound wave *transitions*, not on or off *conditions*. In other words, the transition from on to off or from off to on can be heard as only a slight click; but many of them in quick succession sound like a buzz; in even faster succession, they become actual pitches.

```
0000 C07FOA 00150 CALL 0A7FH ; ACCEPT BASIC VARIABLE
0003 E5 00160 PUSH HL ; STASH BASIC VARIABLE
0004 C1 00170 POP BC ; TRANSFER VAR. TO BC
0005 C5 00180 LOOP PUSH BC ; SAVE OUTER LOOP VAR.
0006 41 00180 LO B,C ; GET MSB FOR TONE
0007 10FE 00200 OJNZ $-0 ; AND DELAY THAT TIME
0008 3A3040 00210 LO A,(4030H) ; GET SCREEN STATUS
000C FS02 00220 OR 2 ; AND SET BIT ONE
000E 03FF 00230 OUT (0FFH),A ; OUTPUT PART OF WAVE
0010 41 00240 LO B,C ; GET VARIABLE AGAIN
0011 10FE 00250 OJNZ $-0 ; DELAY WHILE WAVE HIGH
0013 E6F0 00260 AND 0F0H ; AND SET OTHER BIT HIGH
0015 03FF 00270 OUT (0FFH),A ; TO 00 OTHER WAVE HALF
0017 C1 00280 POP BC ; RESTORE OUTSIDE VALUE
0018 10EB 00280 OJNZ LOOP ; AND LOOP BACK FOR FREQ.
001A CB 00300 RET ; GO BACK TO BASIC
00310 ;
00320 ;
00330 ;
0000 00330 ENO
00000 TOTAL ERRORS
LOOP 0005 00180 00290
```

The single-line BASIC program, short though it may be, cannot move fast enough to produce pleasant tones or dramatic sound effects. For that you must turn to machine language for its speed. Listing 3-(?) presents a complete sound subroutine with ten sample sounds built in.

Listing 3-(?) presents a different sound subroutine which accepts values passed from BASIC.

Again, each program has an advantage. The first allows you to tailor specific sounds and their durations and repetitions with care; the second lets you develop many different sounds easily, directly from BASIC, without altering the machine subroutine already in place.

## What's in a List?

Often it's reassuring to be able to give your program to others without having to worry about gratuitous examination of your code, and finding your own, carefully developed techniques in someone else's product. The easiest thing to do, then, is UN-LIST the program.

Actually, the program still LISTS, but can't be seen; and the program still LLISTS, but uses a whole sheet of paper for each line. In either case, it's a discouragement if you've got some code you like. And protecting just a few lines might give the impression that's all there is to your program – a psychological ploy.

The trick is to add two bytes to the end of the program: the command to 'Home Cursor', and the command to 'Form-Feed'. Here are a few normal program lines:

```
0 REM&&
1 CLS:PRINT:PRINT"CHANGING...":X=17129:REM&&
2 X=PEEK(X)+256*PEEK(X+1):PRINTX"...":REM&&
3 IFX=0THEN10ELSEPOKEX-2,28:POKEX-3,12:GOTO2:REM&&
10 PRINT:PRINT"THIS IS A TEST ":REM&&
20 PRINT"OF A TECHNIQUE TO KILL":REM&&
30 PRINT"THE FEATURE THAT LISTS":REM&&
40 PRINT"OR LLISTS THE PROGRAM.":REM&&
50 PRINT"THE PROGRAM HAS NOW BEEN CHANGED":REM&&
60 PRINT"A COMPLETE LIST FOLLOWS...":REM&&
70 LIST:REM&&
80 REM&&
```

This process uses four bytes per line (colon, REM, two ampersands), and diminishes the usable characters per line by six, but if you have the memory and think you need a little protection, this is a way to go. Delete lines 1, 2 and 3, and save the program. When LIST is commanded, nothing will list on the screen but two REM lines (0 and 80). The program will require nine sheets of paper to LLIST.

```

00100 ; #####
00110 ; BASIC AUTOEXECUTION ROUTINE PATCH FOR LEVEL II BASIC
00120 ; #####
107B 00140 BYTE EQU 107BH ; INTERPRETER CALL A00R.
00150 ;
00160 ; #####
00170 ;
7F00 00180 ORG 7F00H ; RELOCATE IF DESIRED
7F00 2A0440 00180 START LO HL,(4004H) ; GET INTERPRETER PATCH
7F03 22587F 00200 LO (RETURN),HL ; CHAINING PROCESS HERE
7F06 210F7F 00210 LO HL,ENTRY ; ENTRY OF AUTOEXECUTOR
7F09 220440 00220 LO (4004H),HL ; PATCH INTO INTERPRETER
7F0C C3CC06 00230 JP 06CCH ; RETURN TO BASIC READY
00240 ;
00250 ; #####
00260 ; CHECK FOR STATUS OF INTERPRETER (MUST BE AT 105BH)
00270 ; #####
00280 ;
7F0F E3 00280 ENTRY EX (SP),HL ; GET RETURN ADDRESS
7F10 70 00300 LO A,L ; GET LSB INTO A REG.
7F11 FE58 00310 CP 5BH ; CHECK LSB OF 105BH
7F13 2003 00320 JR NZ,NOTROY ; GO OUT IF NOT AT 105B
7F15 7C 00330 LO A,H ; GET MSB INTO A REG.
7F16 FE10 00340 CP 10H ; CHECK MSB OF 105BH
7F18 E3 00350 NOTROY EX (SP),HL ; RETURN STACK POSITION
7F19 C2577F 00360 JP NZ,AWAY ; BEGONE IF NOT 105BH
00370 ;
00380 ; #####
00390 ; COMPARE PRESENT LINE POSITION WITH CLOAD (TOKEN B9)
00400 ; #####
00410 ;
7F1C C07B10 00420 CALL BYTE ; GET NEXT BUFFER CHAR.
7F1F F5 00430 PUSH AF ; SAVE PRESENT ACCUM.
7F20 FEB9 00440 CP 0B9H ; SEE IF CLOAD TOKEN
7F22 2804 00450 JR Z,CLOAD ; SPECIAL ROUTINE IF B9
7F24 F1 00460 POP AF ; RESTORE PRESENT ACCUM.
7F25 28 00470 DEC HL ; RESTORE HL POINTER
7F26 1B2F 00480 JR AWAY ; OUT TO NORMAL MODE
00490 ;
00500 ; #####
00510 ; IF CLOAD TOKEN IS FOUND, EXECUTE CLOAD PROCESS, BUT
00520 ; FIRST INTERCEPT KEYBOARD SCAN (CRUDE WAY OF DOING IT)
00530 ; TO GRAB PROGRAM ON ITS WAY BACK TO A READY CONDITION
00540 ; #####
00550 ;
7F28 F1 00560 CLOAD POP AF ; CLEAR STACK OF AF REG.
7F29 2A1640 00570 LO HL,(4016H) ; GET CURRENT KEYBOARD SCAN
7F2C 22557F 00580 LO (STORE),HL ; SAVE IT FOR A WHILE
7F2F 21427F 00590 LO HL,BYPASS ; GET VALUE OF ROUTINE
7F32 221640 00600 LO (4016H),HL ; PATCH INTO KEYBOARD PLACE
7F35 21E941 00610 LO HL,41E9H ; POINT TO BUFFER LOC'N
7F38 3600 00620 LO (HL),00 ; PLACE END OF LINE CMD.
7F3A 2B 00630 DEC HL ; BUMP HL BACK TO START
7F3B 3EB9 00640 LO A,0B9H ; GET CLOAD VALUE
7F3D 0680 00650 SUB B0H ; STRIP OFFSET VALUE
7F3F C36510 00660 JP 1065H ; BACK TO EXECUTION ROUT.
00670 ;
00680 ; #####
00690 ; CLOAD ABOVE WILL EXECUTE, RESET THE STACK, AND ATTEMPT
00700 ; TO RETURN TO A READY CONDITION. AT THIS POINT, THE
00710 ; KEYBOARD DRIVER INTERCEPT WILL REDIRECT THE PROCESSOR
00720 ; TO THE BYPASS ROUTINE BELOW, WHICH WILL AGAIN SET UP
00730 ; THE BUFFER, REPATCH THE KEYBOARD DRIVER, AND RUN.
00740 ; #####
00750 ;
7F42 2A557F 00760 BYPASS LO HL,(STORE) ; GET BACK KEYBOARD SCAN
7F45 221640 00770 LO (4016H),HL ; PUT BACK INTO PLACE
7F48 21E841 00780 LO HL,41E8H ; GET BUFFER LOCATION
7F48 368E 00790 LO (HL),8EH ; GET RUN COMMAND TOKEN
7F4D 23 00800 INC HL ; BUMP UP BUFFER POS'N
7F4E 3600 00810 LO (HL),00 ; CLOSE OFF THE BUFFER
7F50 2B 00820 DEC HL ; BUMP BACK IN BUFFER...
7F51 2B 00830 DEC HL ; ...TO THE RUN COMMAND
7F52 C35A10 00840 JP 105AH ; AND THEN EXECUTE IT
00850 ;
00860 ; #####
00870 ; THE FOLLOWING FOUR BYTES ARE TEMPORARY KEYBOARD STORAGE
00880 ; #####
00890 ;
7F55 0000 00900 STORE DEFW 0000 ; TEMPORARY TWO-BYTE AREA
7F57 C3 00910 AWAY DEFB 0C3H ; JUMP COMMAND IN PLACE
7F58 7B10 00920 RETURN DEFW 107BH ; ORIGINAL VALUE CHANGES
00930 ;
7F00 00840 END START ; PATCH ROUTINE AT START
00000 TOTAL ERRORS
31717 TEXT AREA BYTES LEFT
AWAY 7F57 00910 00360 00480
BYPASS 7F42 00760 00580
BYTE 107B 00140 00420
CLOAD 7F28 00560 00450
ENTRY 7F0F 00280 00210
NOTROY 7F18 00350 00320
RETURN 7F58 00920 00200
START 7F00 00190 00940
STORE 7F55 00900 00580 00760

```

## Autoexecute BASIC Programs

One of the pleasures of disk operating systems is the ability to load and run programs automatically. This can be done with tape systems as well, simply because all the Level II BASIC operations are organized as subroutines. Any one may be called at any time. To autoexecute a program, then:

1. The SYSTEM command must be entered and the load begun in this mode.
2. The SYSTEM tape must load over its own return point so that it can begin execution automatically.
3. The SYSTEM program loaded must CALL the CLOAD routine, first preparing the stack to return to itself instead of command level.
4. Upon return, the SYSTEM program must prepare the stack once again for return to normal Level II operation, and jump to the RUN routine.

The process is straightforward with one exception: the CLOAD routine is terminal. That is, it forces a return to command level upon completion by clearing out the return address on the stack. It means that a program which might have been written in little more than a dozen bytes must instead play some memory hopscotch first.

Before turning to this loading routine itself, here is a look at the heart of the autorun sequence itself – a mere thirteen bytes! Enter any short BASIC program first, then the routine below:

```

5000 21 EB 41      LO    HL,41EB
5003 36 8E        LO    (HL),8E
5005 23          INC    HL
5006 36 00        LO    (HL),00
500B 2B          DEC    HL
5009 2B          DEC    HL
500A C3 5A 10     JP    105A

```

From BASIC, you can put this program in place with the following lines from command level:

```

X=20480 : POKE X,33 : POKE X+1,232 : POKE X+2,65 :
POKE X+3,54 : POKE X+4,241 : POKE X+5,35 : POKE X+6,54 :
POKE X+7,0 : POKE X+8,43 : POKE X+9,43 : POKE X+10,1B5 :
POKE X+11,90 : POKE X+12,29 : <ENTER>.

```

This simple routine sets the HL register to point to the usual beginning of the keyboard input buffer, puts an 8E (the RUN command value) into that place, bumps the register one place forward, and puts a zero there. The HL register is bumped back to just before the beginning of the keyboard buffer, and the execution routine at 1D5A is entered.

## Machine Language Monitor

```

00100 ; #####
00110 ; MINIATURE MACHINE-LANGUAGE MONITOR DISPLAYING HEX/ASCII
00120 ; #####
06CC 00130 READY EOU 06CCH ; RETURN TO READY INTACT
1078 00140 BYTE EOU 107BH ; ROM READ KEY & TOKENIZE
1997 00150 SYNERR EOU 1997H ; ENTRY POINT TO SN ERROR
00160 ; #####
00170 ; GET REST OF DATA AND CONVERT
0000 00180 OPENER CALL BYTE ; NEXT CHARACTER IN LINE
0003 FE22 00190 CP 22H ; IS IT A QUOTE MARK?
0005 C29719 00200 JP NZ,SYNERR ; OUT TO ERROR IF NOT
0008 E5 00210 PUSH HL ; SAVE LINE POINTER
0009 F0E1 00220 POP IY ; STASH POINTER IN IY
000B C0FB00 00230 CALL XX89 ; CONVERT CHARS TO HEX
000E C0C901 00240 CALL 01C9H ; CLEAR SCREEN (ROM CALL)
0011 184C 00250 JR NEXTB9 ; JUMP PAST SUBROUTINES
00260 ; #####
00270 ; GET 16 SCREEN POSITIONS READY (10H)
0013 7A 00280 CONTNT LO A,0 ; GET ADDRESS LOW BYTE
0014 21403C 00280 LO HL,3C40H ; GET SECOND SCREEN LINE
0017 E6F0 00300 ANO 0F0H ; MASK OUT LOW BITS
0019 C0E200 00310 CALL RRRRS ; ROTATE/DISPLAY ROUTINE
001C 7A 00320 LO A,0 ; GET ADDRESS LOW BYTE
0010 E60F 00330 ANO 0FH ; MASK OUT HIGH BITS
001F C00800 00340 CALL HEXA5C ; CONVERT WORKS TO ASCII
0022 77 00350 LO (HL),A ; DISPLAY THE CHARACTER
0023 23 00360 INC HL ; NEXT SCREEN POSITION
0024 78 00370 LO A,E ; GET HIGH BYTE
0025 E6F0 00380 ANO 0F0H ; MASK OUT LOW BITS
0027 C0E200 00390 CALL RRRRS ; ROTATE/DISPLAY ROUTINE
002A 78 00400 LO A,E ; GET HIGH BYTE AGAIN
002B E60F 00410 ANO 0FH ; MASK OUT LOW BITS
0020 C00800 00420 CALL HEXA5C ; CONVERT HEX TO ASCII
0030 77 00430 LO (HL),A ; DISPLAY THE CHARACTER
0031 21803C 00440 LO HL,3C80H ; GET NEXT SCREEN ROW
0034 0610 00450 LO B,10H ; GET 16 VALUE INTO B
00460 ; #####
00470 ; DISPLAY CONTENTS OF ADDRESS CHOSEN
0036 1A 00480 CONT02 LO A,(0E) ; GET VALUE AT ADDRESS
0037 E6F0 00490 ANO 0F0H ; MASK OUT HIGH BITS
0039 C0E200 00500 CALL RRRRS ; CONVERT/DISPLAY ROUTINE
003C 1A 00510 LO A,(0E) ; GET VALUE AT ADDRESS
0030 E60F 00520 ANO 0FH ; MASK OUT LOW BITS
003F C00800 00530 CALL HEXA5C ; CONVERT CHAR TO ASCII
0042 77 00540 LO (HL),A ; DISPLAY THE CHARACTER
0043 23 00550 INC HL ; GET NEXT SCREEN POSN.
0044 23 00560 INC HL ; GO ONE PLACE MORE
0045 13 00570 INC DE ; GET NEXT ADDRESS LOCN.
0046 10EE 00580 DJNZ CONT02 ; FULL 16 BYTES DISPLAYED
00590 ; #####
00600 ; DISPLAY ASCII VALUES TOO
0048 0610 00610 LO B,10H ; GET 16 TIMES IN B REG.
004A 48 00620 LO C,B ; SAVE IT IN C FOR USE
004B 18 00630 DEC C ; GET NEXT LOWEST ADDRESS
004C 10F0 00640 DJNZ $-1 ; DECREMENT BACK TO START
004E 41 00650 LO B,C ; GET 16 TIMES IN B AGAIN
004F 21C03C 00660 LO HL,3C00H ; GET NEXT LINE OF SCREEN
0052 1A 00670 BBBA LO A,(0E) ; GET CONTENTS OF ADDRESS
0053 77 00680 LO (HL),A ; DISPLAY EXACTLY AS IS
0054 23 00690 INC HL ; GET NEXT SCREEN LOCN.
0055 23 00700 INC HL ; GET NEXT AFTER THAT
0056 23 00710 INC HL ; VISUALLY MATCHES HEX
0057 13 00720 INC DE ; GET NEXT ADDRESS TO SEE
0058 10FB 00730 DJNZ BBBA ; DO IT FOR 16 ADDRESSES
005A 41 00740 LO B,C ; GET 16 INTO B AGAIN
005B 18 00750 DEC DE ; GO BACK TO PREVIOUS
005C 10F0 00760 DJNZ $-1 ; AND BACK TO BEGINNING
005E C9 00770 RET ; DONE WITH DISPLAY ROUT.
00780 ; #####
00790 ; SCAN KEYBOARD FOR EDIT SEQUENCE
005F C01300 00800 NEXTB9 CALL CONTNT ; FIND WHICH KEYS PRESSED
00810 ; #####
00820 ; SCAN KEYBOARD FOR BREAK, ARROWS
0062 3A403B 00830 EDITOR LO A,(3B40H) ; GET BREAK, ARROWS ROW
0065 FE04 00840 CP 4 ; IS IT BREAK KEY?
0067 2006 00850 JR NZ,ARROW ; IF NOT TEST FOR ARROW
0069 F0E5 00860 PUSH IY ; ELSE RETRIEVE LINE PTR.
006B E1 00870 POP HL ; SWITCH BACK INTO HL
006C C3CC06 00880 JP READY ; BACK TO BASIC READY
006F FE10 00890 CP 10H ; BEGIN ARROW COMPARES
0071 2007 00900 JR NZ,AAAA ; GO IF NOT DOWN ARROW
0073 0610 00910 LO B,10H ; GET 8 READY WITH 16
0075 18 00920 DEC DE ; GO BACK IN MEMORY
0078 10F0 00930 DJNZ $-1 ; DO IT FOR 16 TIMES
007B 184B 00940 JR STNORO ; DONE NOW; GO OUT
007A FE08 00950 AAAA CP B ; CHECK IF UP ARROW
007C 2007 00960 JR NZ,AAAB ; GO OUT IF NOT UP ARROW
007E 0610 00970 LO B,10H ; GET 16 PLACES READY
0080 13 00980 INC DE ; GET NEXT MEMORY LOCN.
0081 10F0 00990 DJNZ $-1 ; DO IT 16 TIMES IN ALL
0083 1840 01000 JR STNORO ; DONE NOW; GO OUT
0085 FE20 01010 AAAB CP 20H ; CHECK IF LEFT ARROW
0087 2003 01020 JR NZ,AAAC ; GO OUT IF NOT LEFT
0089 18 01030 DEC DE ; GET PREVIOUS MEM. LOCN.
008A 1839 01040 JR STNORO ; DONE NOW; GO OUT
008C FE40 01050 AAAC CP 40H ; CHECK IF RIGHT ARROW
008E 2003 01060 JR NZ,AAAD ; GO OUT IF NOT RIGHT
0090 13 01070 INC DE ; GET NEXT MEMORY LOCN.
0091 1832 01080 JR STNORO ; DONE NOW; GO OUT
01090 ; #####
01100 ; GET FIFTH LINE OF SCREEN AND DISPLAY CHOSEN EDITING

```

Listing Continued . . .

The routine at 1D5A bumps the HL register forward, evaluates the byte (finding 8E = RUN), then looks for a possible line number to execute. Finding a zero means the command ends there, and so a simple RUN routine is entered. Here's how to try it out once you have it in place:

```

SYSTEM <ENTER>
/20480 <ENTER>

```

The BASIC program you had entered earlier should now run just as any other BASIC program might. So the idea is to make this autorun routine the heart of the area that the CLOAD might make its way back to.

Listing 3-(?) presents a machine language program which must precede any program to be autoexecuted. It follows the rules above by taking over control of the computer, placing a patch into the keyboard scan in order to intercept the terminal CLOAD routine's return to BASIC, and directing the computer to the usual CLOAD routine. When CLOAD gets back into BASIC, it will present a 'READY' and begin to scan the keyboard. It will, however, never get there.

Instead, the intercept now patched in place will redirect the computer to a short routine also present in the keyboard input buffer area. This routine restores the original plundered keyboard return address, and executes the automatic RUN routine. The autoloader remnants in the keyboard buffer are no longer needed, and will be wiped out at the next keyboard input of any kind.

To use this program, assemble it and save it at the beginning of each in a batch of tapes. Use these tapes to CSAVE any programs you wish to autoexecute. Whenever you wish to run one of these programs, type . . .

```

>SYSTEM <ENTER>
*? AUTO <ENTER>

```

. . . and the program will load, acting as if a normal CLOAD were in action, but immediately beginning execution of the BASIC routine.

## Machine Language Monitor

It can be very frustrating when you need to make some quick alterations to memory, or when you need to install a short machine language program. The options are few: load a decimal-to-hex conversion program and enter the code; convert the values to hex by hand and POKE them in place; write the code into a short

## Continued Listing

```

0083 210030 01110 AAOO LO HL,3000H ; FIFTH LINE ON SCREEN
0098 365F 01120 LO (HL),5FH ; GET CURSOR CHARACTER
0098 23 01130 INC HL ; NEXT SCREEN LOCATION
0088 365F 01140 LO (HL),5FH ; GET CURSOR CHARACTER
0088 28 01150 DEC HL ; BACK TO FIRST LOCN.
008C 0602 01160 LO B,2 ; GET 2 TRIES INTO B
008E 05 01170 AAAE PUSH OE ; SAVE MEMORY LOCATION
009F E5 01180 PUSH HL ; SAVE SCREEN LOCATION
00A0 C04900 01190 CALL 0049H ; BASIC'S KEYBOARD SCAN
00A3 E1 01200 POP HL ; RESTORE SCREEN LOCN.
00A4 01 01210 POP OE ; RESTORE MEMORY LOCN.
00A5 FE47 01220 CP 47H ; CHECK IF ALPHA HEX
00A7 308B 01230 JR NC,EDITOR ; GO OUT IF NOT ALPHA HEX
00A9 FE30 01240 CP 30H ; CHECK IF NUMERIC HEX
00A8 38B5 01250 JR C,EDITOR ; OUT IF NOT NUMERIC HEX
00A0 FE3A 01260 CP 3AH ; CHECK IF OV NUMERIC
00AF 3804 01270 JR C,AAAF ; CHECK NEXT IF IN RANGE
00B1 FE40 01280 CP 40H ; CHECK IF OV ALPHAHEX
00B3 38A0 01290 JR C,EDITOR ; OUT IF OV ALPHA HEX
00B6 77 01300 AAAF LO (HL),A ; PLACE CHAR ON SCREEN
00B6 23 01310 INC HL ; GET NEXT SCREEN LOCN.
00B7 10E5 01320 OJNZ AAAE ; GO GET ANOTHER CHAR.
01330 ; #####
01340 ; CONVERT CHOSEN DATA TO HEX
008B 28 01350 DEC HL
008A C0C000 01360 CALL ASCHEX
008D 4F 01370 LO C,A
008E 28 01380 DEC HL
008F C0EC00 01390 CALL LLLLS
00C2 81 01400 ADD A,C
01410 ;
01420 ; PUT NEW BYTE IN PLACE
00C3 12 01430 LO (DE),A
00C4 13 01440 INC DE
01450 ;
01460 ; DISPLAY REVISED LINE OF DATA
00C5 C01300 01470 STNORD CALL CONTNT
00C8 C0F400 01480 CALL DELAY
00C8 18B5 01490 JR EDITOR
01500 ;
01510 ; ASCII TO HEXADECIMAL CONVERSION
00C0 7E 01520 ASCHEX LO A,(HL)
00CE FE40 01530 CP 40H
00D0 3003 01540 JR NC,NEXTBB
00D2 0630 01550 SUB 30H
00D4 C9 01560 RET
00D5 0637 01670 NEXTBB SUB 37H
00D7 C9 01580 RET
01590 ;
01600 ; HEXADECIMAL TO ASCII CONVERSION
00D8 FE0A 01610 HEXASC CP 0AH
00DA 3003 01620 JR NC,NEXT96
00DC C630 01630 ADD A,30H
00DE C9 01640 RET
00DF C637 01650 NEXT96 ADD A,37H
00E1 C9 01660 RET
01670 ;
01680 ; RIGHT ROTATES FOR CONVERSIONS
00E2 0F 01690 RRRS RLCA
00E3 0F 01700 RRRS RLCA
00E4 0F 01710 RRRS RLCA
00E5 0F 01720 RRRS RLCA
00E8 C00B00 01730 CALL HEXASC
00EB 77 01740 LO (HL),A
00EA 23 01750 INC HL
00EB C9 01760 RET
01770 ;
01780 ; LEFT ROTATES FOR CONVERSION
00EC C0C000 01790 LLLS CALL ASCHEX
00EF 07 01800 RLCA
00F0 07 01810 RLCA
00F1 07 01820 RLCA
00F2 07 01830 RLCA
00F3 C8 01840 RET
01850 ;
01860 ; DELAY FOR SCREEN DISPLAYS
00F4 010020 01870 DELAY LO BC,2000H
00F7 C06000 01880 CALL 00B0H
00FA C9 01890 RET
01800 ;
01810 ; GET/CONVERT ASCII FROM BUFFER
00F8 0604 01830 XX98 LO B,4
00F0 C07B10 01840 SSSS CALL BYTE
0100 F6 01850 PUSH AF
0101 10FA 01860 OJNZ SSSS
0103 F1 01870 POP AF
0104 77 01880 LO (HL),A
0105 C0C000 01890 CALL ASCHEX
0108 5F 02000 LO E,A
0109 F1 02010 POP AF
010A 77 02020 LO (HL),A
010B C0EC00 02030 CALL LLLLS
010E 83 02040 ADD A,E
010F 5F 02050 LO E,A
0110 F1 02060 POP AF
0111 77 02070 LO (HL),A
0112 C0C000 02080 CALL ASCHEX
0115 67 02090 LO 0,A
0116 F1 02100 POP AF
0117 77 02110 LO (HL),A

```

BASIC program that does the work. None of these are satisfactory. Ideally, a machine language monitor is the tool to use.

But there are disadvantages to the monitors currently available. Many are too long, and are part of other, lengthier programs. Others overlap resident BASIC programs. And none make available ASCII representations as well as BASIC graphics characters. The short monitor presented in this section provides the latter, can be executed from BASIC (using the patch table presented earlier), and sits wherever in memory you would like.

It consists of a few major sections: The first clears the display, presents the requested address, displays the hex contents of that address and the sixteen following, displays the ASCII or graphics values of that address and the sixteen following, and presents a cursor for hex code entry. The second section searches the keyboard for a valid hex character, displays the character, waits for another, displays that, and advances the address and display.

The second section also searches the keyboard for the arrows, and advances the display (a) one place forward on a right arrow; (b) one place back on a left arrow; (c) sixteen places forward on an up arrow; and (d) sixteen places back on a down arrow. Last of all it searches for the (BREAK) key, which returns it to BASIC.

This monitor, as with all the BASIC-transparent programs presented in this book, must be executed by using the special command patch table (see Listing (?)-(?) ). The command used in this table is /OPEN"NNNN", where NNNN is the address to be opened for examination (in hex).

## Undoing NEW

This is a much easier task than the Level II manual would have you believe. When the NEW command is entered, the program remains in place, *completely unchanged!* The only alteration is that the end-of-BASIC-program pointer in memory has been changed to the beginning of the program. Hence, the computer believes that the program has a total length of zero.

But the old end-of-program information is still intact elsewhere, and can be found very easily. In fact, to restore a program you have actually

Listing Continued . . .

## Resetting MEMORY SIZE?

### Continued Listing

```

0118 COEC00 02120      CALL  LLLLS
0118 B2      02130      A00    A,D
011C 57      02140      LO      0,A
0110 C9      02150      RET
08CC         02160      EMO     READY
0000 TOTAL ERRORS
AAAA 007A 00950 00900
AAAB 0085 01010 00960
AAAC 008C 01050 01020
AAAD 0093 01110 01060
AAAE 009E 01170 01320
AAAF 0085 01300 01270
ARROW 006F 00890 00850
ASCHEX 00C0 01520 01360 01790 01990 02080
88BA 0052 00670 00730
BYTE 1078 00140 00180 01540
CONT02 0036 00480 00580
CONTNT 0013 00280 00800 01470
DELAY 00F4 01870 01480
EDITOR 0062 00830 01230 01250 01250 01490
HEXASC 0008 01610 00340 00420 00530 01730
LLLLS 00EC 01790 01390 02030 02120
NEXT96 000F 01650 01620
MEXT98 0005 01570 01540
MEXT99 005F 00800 00250
OPENER 0000 00180
READY .06CC 00130 00880 02160
RRRHS 00E2 01690 00310 00390 00500
SSSS 00F0 01940 01860
STNOR0 00C5 01470 00940 01000 01040 01080
SYNERR 1897 00150 00200
XX99 00F8 01930 00230

```

```

00100 ; #####
00110 ; ONLY A SHORT ROUTINE IS NECESSARY TO DETERMINE WHERE
00120 ; A BASIC PROGRAM WAS BEFORE ITS POINTERS WERE RESET. THE
00130 ; BEGINNING OF PROGRAM POINTER MUST BE RESET, THE LINE
00140 ; NUMBERS RUN THROUGH, THE EMO OF PROGRAM POINTER RESET,
00150 ; THE STACK POINTER RESET, THE SCREEN CLEARED, AND THE
00160 ; CLEAR (RESET VARIABLES AND PROGRAM CONDITIONS) EXECUTED
00170 ; #####
00180 ; NOTE THAT THIS ROUTINE IS ALSO CALLED FROM BASIC USING
00190 ; THE FORMAT /NEW. THE CUSTOM INTERPRETER IS EMPLOYED.
00200 ; #####
00210 ;
0000 E058A440 00220 REMEW LO DE,{40A4H} ; GET START OF PRGRM PTR
0004 3EFF 00230 LO A,0FFH ; GET FF RESETTNG CODE
0006 12 00240 LO {0E},A ; PLACE AT PROGRAM START
0007 C0FC1A 00250 CALL 1AFCH ; GO THRU ALL LINES TILL
; EMO OF PRGRM 00 F000
000A 23 00260 ; HL MOVED JUST PAST PRGM
000B 22FB40 00270 IMC HL ; SIMPLE VARIABLE POINTER
000E E07BE840 00280 LO {40FBH},HL ; RESET STACK TO NORMAL
0012 C0C901 00290 LO SP,{40EBH} ; CLEAR THE SCREEN NOW
0015 C06118 00300 CALL 01C9H ; CLEAR ALL THE POINTERS
0018 C3CC06 00310 CALL 1861H ; BACK TO BASIC "READY"
; 06CCH
00330 ;
00340 ; #####
0000 00350 EMO
00000 TOTAL ERRORS
RENEW 0000 00220

```

```

00100 ; #####
00110 ; RAM CHECKING PROGRAM - FIRST VERSION WITHOUT DISPLAY -
00120 ; EXHAUSTIVELY CHECKS EACH TRIO OF RAM LOCATIONS WITHOUT
00130 ; DISPLAYING CURRENT ADDRESS; BAD ADDRESSES ONLY SHOWN.
00140 ; #####
00150 ; NOTE THAT THIS PROGRAM IS * SELF-RELOCATING * AND MUST
00160 ; BE ASSEMBLED AT THE ORIGIN ADDRESS SPECIFIED BELOW.
00170 ; #####
5000 00180 ORG 5000H ; DO NOT CHANGE ORIGIN!
00200 ;
00210 ; #####
00220 ; SUBROUTINE TO CLEAR THE SCREEN WITH NORMAL SPACES
00230 ; #####
00240 ;
5000 21003C 00250 LO HL,3C00H ; BEGINNING OF VIDEO
5003 11013C 00260 LO DE,3C01H ; DESTINATION OF SPACE
5006 01FF03 00270 LO BC,03FFH ; SPACES ON SCREEN
5008 3620 00280 LO [HL],20H ; DISPLAY BLANK SPACE
5008 E080 00290 LOIR ; CLEAR THE SCREEN
00300 ;
00310 ; #####
00320 ; SET UP MIDDLE OF PROGRAM POINTER, DISPLAY STAR TO SHOW
00330 ; PROGRAM IS WORKING, SET UP THE STACK POINTER IN RELA-
00340 ; TIONSHIP TO THE IX REGISTER, AND SPECIFY START OF TEST
00350 ; #####

```

wiped out, you need to invoke a few ROM routines and restore the beginning-of-program pointer.

Although variables are cleared in this process, the program is totally restored. If you wish to make this a part of a transparent operating system using the interpreter patch presented earlier in this chapter, Listing 3-(?) presents a complete routine. Enter /NEW, and the lost program reappears.

One warning is in order: if before restoring the program you cause a ?SN ERROR, the computer will jumble up the first part of the program, mess around with some other memory pointers, and the program will *really* be lost.

## Resetting MEMORY SIZE?

The size of BASIC memory available can also be changed from BASIC itself, because it too is simply stored in a two-byte pointer in the RAM patch area. New values may be POKed into place, so long as they meet two conditions:

1. The new value must be within the range of actual memory available.
2. It must not dip below the top of an existing BASIC program already in memory.
3. If no program is in memory, it must not dip below address 4414.

Here's how to do it. Convert the desired new memory size to split decimal with this formula:

```

X = NNNNN      <ENTER>
Y = FIX (X/256) <ENTER>
Z = X - Y * 256 <ENTER>
PRINT Z,Y      <ENTER>

```

The value of Z is the least significant byte of the new memory size, Y is the most significant byte. Now:

```
POKE 16561,Z : POKE 16562,Y : CLEAR50
```

The new memory size has been set, and 50 bytes are cleared for string space, as usual. If you haven't followed the rules about legitimate memory sizes, expect a fast system crash.

Listing Continued . . .

## Continued Listing

```

00360 ;
5000 00218050 00370 ; LO IX,5080H ; MIDDLE OF TEST PROGRAM
5011 3AFE3F 00380 ; LO A,{3FFE} ; BOTTOM RIGHT OF SCREEN
5014 EEOA 00390 ; XOR 0AH ; ALTERNATE SPACE & STAR
5016 32FE3F 00400 ; (3FFE),A ; DISPLAY IT ON SCREEN
5018 00F9 00410 ; LO SP,IX ; START STACK POINTER
5018 067F 00420 ; LO B,7FH ; NUMBER OF MOVES FOR 6P
5010 33 00430 ; INC SP ; SP = SP + 1
501E 10F0 00440 ; OJNZ $-1 ; DO IT 127 TIMES
5020 210060 00450 ; LO HL,6000H ; BEGINNING OF TEST AREA

00460 ;
00470 ; #####
00480 ; TEST BEGINS WITH THE VALUE OF ZERO, AND PROCEEDS TO
00490 ; VALUE FF. EACH VALUE IS WRITTEN IN TURN TO EACH OF A
00500 ; TRIO OF MEMORY LOCATIONS IN ORDER TO DETERMINE THEIR
00510 ; EFFECT ON EACH OTHER AS VALID ELECTRONIC STORAGE CELLS.
00520 ; #####
00530 ;

5023 AF 00640 ; XOR A ; CLEAR ACCUMULATOR
5024 77 00650 ; LO (HL),A ; PLACE VALUE IN MEMORY
5025 4F 00660 ; LO C,A ; PLACE VALUE IN C
5026 F5 00670 ; PUSH AF ; SAVE VALUE IN ACCUM.
5027 7B 00680 ; LO A,C ; GET VALUE FROM C
5028 77 00690 ; LO (HL),A ; PLACE VALUE IN MEMORY
5028 23 00600 ; INC HL ; INCREMENT TO NEXT MEM
502A 7C 00610 ; LO A,H ; GET MSB OF ADDRESS
5028 FE80 00620 ; CP 80H ; IS MEM AT 8000 YET?
00630 ; ; NOTE THAT THE VALUE
00640 ; ABOVE IS FOR 16K; USE
00650 ; CO FOR 32K, OR FOR 48K

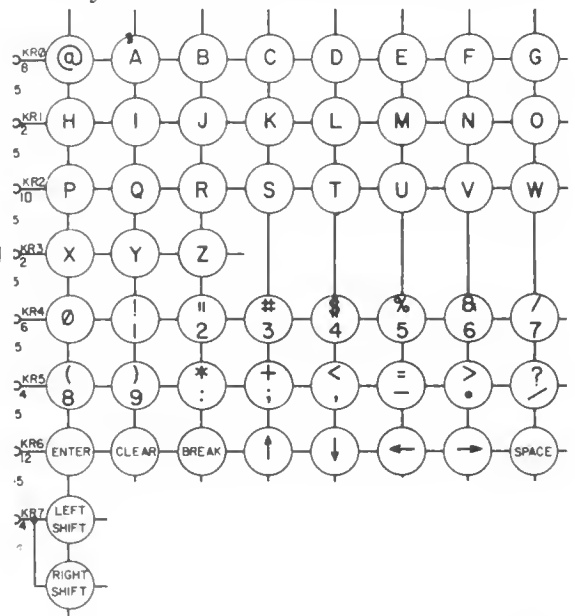
5020 2830 00660 ; JR Z,$+32H
502F E5 00670 ; PUSH HL ; GAVE MEMORY VALUE
5030 00E5 00680 ; PUSH IX ; SAVE MIDDLE OF PROGRAM
5032 E1 00690 ; POP HL ; GET POSITION INTO HL
5033 8C 00700 ; CP H ; CHECK AGAINST 80 VALUE
5034 2828 00710 ; JR Z,$+28H ; RELOCATE IF 00NE
5036 E1 00720 ; POP HL ; RESTORE ORIGINAL VALUE
5037 F1 00730 ; POP AF ; RESTORE ORIGINAL TEST
5038 77 00740 ; LO (HL),A ; PUT VALUE INTO MEMORY
5039 28 00750 ; DEC HL ; BACK TO ORIGINAL LOC'N
503A 28 00760 ; DEC HL ; BACK TO ONE BEFORE IT
5038 77 00770 ; LO (HL),A ; PUT VALUE INTO MEMORY
503C 47 00780 ; LO B,A ; SAVE VALUE IN B REG.
5030 7E 00790 ; LO A,(HL) ; GET VALUE AT LOC'N HL
503E 88 00800 ; CP B ; CHECK AGAINST B VALUE
503F 2805 00810 ; JR Z,$+7 ; GO ON IF IT CHECKS OKAY
5041 1618 00820 ; LO 0,18H ; GET VALUE OF REL. SUB.
5043 05 00830 ; PUSH OE ; SAVE VALUE ON STACK
5044 1818 00840 ; JR $+10H ; JUMP TO SUBROUTINE
5046 23 00850 ; INC HL ; GET ORIGINAL TEST POS'N
5047 23 00860 ; INC HL ; GO ONE BEYOND IT
5048 7E 00870 ; LO A,(HL) ; GET VALUE AT THAT POS'N
5049 88 00880 ; CP B ; CHECK AGAINST B REG.
504A 2805 00890 ; JR Z,$+7 ; GO ON IF MEMORY OKAY
504C 1610 00900 ; LO 0,10H ; GET JUMP FOR REL. SUB.
504E 05 00910 ; PUSH OE ; SAVE VALUE ON STACK
504F 1810 00920 ; JR $+12H ; JUMP TO SUBROUTINE
5051 28 00930 ; DEC HL ; BACK TO ORIGINAL POS'N
5052 34 00940 ; INC (HL) ; INCREMENT VALUE IN MEM
5053 0C 00950 ; INC C ; INCREMENT TEST VALUE
5054 7E 00960 ; LO A,(HL) ; GET VALUE IN MEMORY
5055 47 00970 ; LO B,A ; SAVE VALUE IN B REG.
5056 FE00 00980 ; CP 00 ; CHECK IF 256 BYTES DONE
5058 20CC 00990 ; JR NZ,$-32H ; LOOP BACK AND CONTINUE
505A 23 01000 ; INC HL ; GET NEXT MEMORY VALUE
505B 0E00 01010 ; LO C,0 ; RESET TEST VALUE TO 0
5050 18C7 01020 ; JR $-37H ; LOOP BACK FOR NEXT TEST
505F 1864 01030 ; JR $+66H ; REL.SUB. STEPPING STONE
01040 ;
01050 ; #####
01060 ; SUBROUTINE BELOW IS ENTERED WHEN A 8AD MEMORY LOCATION
01070 ; HAS BEEN DETERMINED. IT CONVERTS HEX VALUES TO ASCII
01080 ; AND DISPLAYS THEM ON THE SCREEN. NOTE THE RELATIVE
01090 ; SUBROUTINE ENTRY AND EXIT METHOD USING THE 0 REGISTER.
01100 ; #####
01110 ;

5061 7C 01120 ; LO A,H ; GET VALUE FROM H REG.
5062 EBF0 01130 ; AND 0F0H ; MASK OFF LOW BITS
5064 0F 01140 ; RRCA ; ROTATE RIGHT FOR CONV.
5065 0F 01150 ; RRCA ; ... SOME MORE ...
5068 0F 01160 ; RRCA ; ... AND SOME MORE ...
5067 0F 01170 ; RRCA ; ... UNTIL IT'S DONE.
5068 1622 01180 ; LO 0,22H ; GET VALUE FOR REL. SUB.
506A 1822 01190 ; JR $+24H ; AND JUMP TO SUBROUTINE
506C 7C 01200 ; LO A,H ; GET VALUE FROM H REG.
506D E60F 01210 ; AND 0FH ; MASK OUT HIGH BITS
506F 1618 01220 ; LO 0,18H ; GET VALUE FOR REL. SUB.
5071 1818 01230 ; JR $+10H ; AND JUMP TO SUBROUTINE
5073 70 01240 ; LO A,L ; GET VALUE FROM L REG.
5074 E6F0 01250 ; AND 0F0H ; MASK OUT LOW BITS
5076 0F 01260 ; RRCA ; ROTATE FOR CONVERSION
5077 0F 01270 ; RRCA ; ... SOME MORE ...
5078 0F 01280 ; RRCA ; ... SOME MORE ...
5078 0F 01280 ; RRCA ; ... AND IT'S DONE.
507A 1610 01300 ; LO 0,10H ; GET VALUE FOR REL. SUB.
507C 1810 01310 ; JR $+12H ; AND JUMP TO SUBROUTINE
507E 70 01320 ; LO A,L ; GET VALUE FROM L REG.
507F E60F 01330 ; AND 0FH ; MASK OUT HIGH BITS
5081 1609 01340 ; LO 0,9 ; GET VALUE FOR REL. SUB.
5083 1808 01350 ; JR $+08H ; AND JUMP TO SUBROUTINE
5085 01 01360 ; POP OE ; RESTORE VALUE TO OE
5086 3E03 01370 ; LO A,003H ; GET RETURN POS'N VALUE
    
```

## Peek that Keyboard

One of the handiest functions for fast-running, convenient BASIC programs is the INKEY\$ command. This allows the latest keystroke to be transferred to a program variable, where it can then be evaluated.

For action games, BASIC word processors and other speed-conscious programs, INKEY\$ can be too long because it requires considerable evaluation and juggling of memory space. There is a faster way, and it involves examining the keyboard's memory contents directly. Below is the keyboard matrix:



You can see that various memory locations refer to various rows of keys. Depending on the key pressed, different values will be discovered by PEEKing. For example, A=PEEK(14400) returns a value of 1 if ENTER is pressed, and a value of 2 if CLEAR is pressed, 4 if BREAK is pressed, 8 for down arrow, 16 for up arrow, 32 for left arrow, 64 for right arrow, and 128 for a space. If only a few keys are being sought – perhaps the arrow in an action game – a very tight loop can be constructed that is many times faster than INKEY\$:

```

10 A = PEEK(14400) : IF A=0 THEN 10
20 IF A=8 THEN 100 : IF A=16 THEN 200
30 IF A=32 THEN 300 : IF A=64 THEN 400
40 GOTO 10
    
```

For top-speed operation, this should all appear on one line; try it in comparison with testing for CHR\$(8) or CHR\$(9), etc., etc., to determine the arrows.

Listing Continued...

## Peek That Keyboard

### Continued Listing

```

5088 92      01380      SUB      0      ; SUBTRACT RETURN OIFF.
5089 007700  01390      LO      (IX+00H),A ; AND MAKE JR OPERAND
508C 18FF    01400      JR      $+1      ; IRRELEVANT OPERAND
508E 05      0141U     PUSH     OE      ; SAVE VALUE ON STACK
508F F5      01420     PUSH     AF      ; SAVE VALUE ON STACK
5090 11003C  01430      LO      0E,3C00H ; GET TOP LEFT OF SCREEN
5093 1A      01440      LO      A,(OE)   ; GET VALUE ON SCREEN
5094 FE20    01450      CP      20H     ; IS IT A SPACE NOW?
5096 2B03    01460      JR      Z,$+5    ; IF SO, GO AHEAD SOME
5098 13      01470      INC      OE      ; INCREMENT SCREEN POS'N
5099 18F8    01480      JR      $-6     ; AND GO AHEAD PAST TEST
509B 78      01490      LO      A,E     ; GET VALUE OF SCREEN
509C FE04    01500      CP      04      ; IS IT 4 POS'NS OVER?
509E 200A    01510      JR      NZ,$+0CH ; IF NOT, GO ON AHEAD
50A0 3E20    01520      LO      A,20H   ; IF SO GET A SPACE READY
50A2 18      01530      DEC      OE      ; GO BACK SOME....
50A3 12      01540      LO      (OE),A  ; AND FILL WITH A SPACE
50A4 18      01550      DEC      OE      ; AND BACK SOME MORE....
50A5 12      01560      LO      (OE),A  ; AND INSERT ANOTHER ONE
50A6 18      01570      DEC      OE      ; AND BACK A BIT MORE....
50A7 12      01580      LO      (OE),A  ; AND STASH ANOTHER SPACE
50A8 18      01590      DEC      OE      ; AND BACK ONE MORE TIME
50A9 12      01600      LO      (OE),A  ; AND STUFF A SPACE THERE
50AA F1      01610      POP      AF      ; RESTORE VALUE TO AF
50AB FE0A    01620      CP      0AH     ; IS VALUE LESS THAN 10?
50AD 3004    01630      JR      NC,$+6    ; IF LESSER, THEN JUMP
50AF C630    01640      AOO      A,30H   ; CONVERT HEX TO ASCII
50B1 1802    01650      JR      $+4     ; AND GO ON PAST THE REST
50B3 C637    01660      AOO      A,37H   ; CONVERT HEX TO ASCII
50B5 12      01670      LO      (OE),A  ; AND STASH ON THE SCREEN
50B6 12      01680      ;
50B7 06FF    01690      ; #####
50B8 10FE    01700      ; SHORT ROUTINE BELOW USED AS DELAY AFTER LOC'N DISPLAY.
50B9 C1      01710      ; #####
50BA 01      01720      ;
50BB 01      01730      PUSH     BC      ; SAVE VALUE IN BC REG
50BC 01      01740      LO      B,0FFH   ; GET DELAY VALUE
50BD 01      01750      OJNZ    $-0      ; AND DELAY JUST A LITTLE
50BE 01      01760      POP      BC      ; RESTORE BC VALUE
50BF 01      01770      POP      OE      ; RESTORE OE VALUE
50C0 01      01780      LO      A,0C9H   ; GET VALUE TO RETURN
50C1 92      01790      SUB      0      ; SUBTRACT JUMP OFFSET
50C2 007744  01800      LO      (IX+44H),A ; PLACE AS JR OPERAND
50C3 18FF    01810      JR      $+1      ; IRRELEVANT OPERAND
50C4 12      01820      ;
50C5 00E5    01830      ; #####
50C6 00E5    01840      ; PROGRAM RELOCATION ROUTINE. HL AND OE REGISTERS ARE
50C7 E1      01850      ; LOADED FROM THE IX REGISTER, AND MODIFIED BY EXCLUSIVE-
50C8 01      01860      ; ORING WITH A KNOWN VALUE IN A. THUS, A NEW PROGRAM
50C9 7A      01870      ; BEGINNING CAN BE DETERMINED, INTERNAL TEST POSITIONS
50CA 01      01880      ; CAN BE MODIFIED, AND THE IX AND SP POINTERS RESET.
50CB 7A      01890      ; #####
50CC EE20    01900      ;
50CD 57      01910      PUSH     IX      ; SAVE PROGRAM POSITION
50CE 05      01920      PUSH     IX      ; SAVE PROGRAM POSITION
50CF E1      01930      POP      HL      ; TRANSFER TO HL REG.
50D0 7A      01940      POP      OE      ; TRANSFER TO OE REG.
50D1 7A      01950      LO      A,0      ; GET VALUE FROM 0 REG.
50D2 05      01960      XOR      20H     ; TRANSFER TO HIGH MEM.
50D3 07      01970      LO      0,A     ; PUT BACK IN 0 REG.
50D4 05      01980      PUSH     OE      ; AND STASH ON STACK
50D5 00680    01990      LO      B,80H   ; DEC. TO PROGRAM START
50D6 28      02000      DEC      HL      ; AND BEGIN DECREMENTING
50D7 18      02010      DEC      DE      ; FOR BOTH THE REGISTERS
50D8 10FC    02020      OJNZ    $-2      ; UNTIL IT'S ALL 00NE
50D9 01FF0D  02030      LO      BC,0DFFH ; AND GET READY TRANSFER
50DA ED80    02040      LOIR     ; AND THEN DD IT!
50DB DDE1    02050      POP      IX      ; RESTORE NEW VALUE
50DC 3E20    02060      LO      A,20H   ; VALUE TO MDD. ADDRESSES
50DD 00AEA2  02070      XOR      (IX+0A2H) ; MODIFY IX+A2 ADDRESS
50DE 0D77A2  02080      LO      (IX+0A2H),A ; AND STORE IT IN PLACE
50DF 3EF0    02090      LO      A,0000H ; VALUE TO MDD. ADDRESSES
50E0 01      02100      ; CHANGE OPERAND ABOVE TO
50E1 01      02110      ; 80 FOR 32K MEM TEST AND
50E2 01      02120      ; TO 20 FOR 48K MEM TEST
50E3 01      02130      ; MODIFY IX+AC ADDRESS
50E4 00EAC    02140      XOR      (IX+0ACH),A ; AND STORE IT IN PLACE
50E5 D077AC  02150      PUSH     IX      ; STORE THE PROGRAM PTR.
50E6 E1      02160      POP      HL      ; AND TRANSFER IT TO HL
50E7 2E11    02170      LO      L,11H   ; SET LSB OF HL REGISTER
50E8 E9      02180      JP      (HL)     ; JUMP TO PROGRAM START!
50E9 12      02190      ;
50EA 02200    02200      ; #####
50EB 02210    02210      END      50D0H
50EC 00000   02210      TOTAL ERRORS

```

There is yet another use for this PEEK function. As noted in Chapter 2, location 387F reveals if any key is pressed at all. This can be used in a BASIC text editor, for example, and is much faster than INKEY\$ to check for keypressing:

```
10 A = PEEK(14463) : IF A=0 THEN 10
```

This is one of the fastest keystroke-detectors in BASIC, and from there a group of PEEKs could be done, checking the most-used rows of characters first. It's both a matter of taste and programming skill whether INKEY\$ or PEEK is used for keyboard input, but in some situations (such as testing for carriage returns and other control codes), PEEK is the winner.

KEYBOARD PEEK POSITIONS								
DATA:	1	2	4	8	16	32	64	128
ADDRESS:								
14337	@	A	B	C	D	E	F	G
14338	H	I	J	K	L	M	N	O
14340	P	Q	R	S	T	U	V	W
14344	X	Y	Z					
14352	0	1	2	3	4	5	6	7
14368	8	9	:	*	+	<	=	>
14400	ENT	CLR	8KR	UPAR	DNAR	LFTAR	RTAR	SPACE
14464	SHIFT							

Figure 3-(?) is a chart of all the characters and their respective PEEK positions and data returned. Remember, when more than one key is depressed at a time, the row of the data at that address is returned. For example, if PEEK (14400) returns 129, then both ENTER and SPACE are being held down. This is significant because INKEY\$ returns only the last character pressed, and there is no real way of getting to multiple characters. Try this:

```

10 A = PEEK (14400)
20 IF A = 129 THEN 100
30 PRINT "PRESS ENTER AND CLEAR"
40 GOTO 10
100 PRINT"00NE!"

```

Simple as this may appear, it offers an 'out' for programs that need special, unusual input for some functions. It can be used as the sort of double-interlock protection switch found on industrial machinery to keep both hands out of blades and moving parts.

Furthermore, when the keyboard is PEEKed, the values returned can be changed and mutated at will – for example, a BASIC letter-writer could have two easily written routines to make the keyboard the standard 'QWERTY' type or changing it to the faster Dvorak type. Of course, even with PEEK statements, BASIC is not likely to keep up with that sort of speed typing!

## Make-'Em-Sweat Memory Test

Many memory tests are available for testing the dynamic memory in the TRS-80 and expansion interface, including the simple test done by the computer itself on power-up (see Supplement to Chapter 1). Most have a significant disadvantage: they are *software* tests rather than *electrical* tests. Certainly, since memory involves operating software, a software test seems to be a logical solution.

On the other hand, memories are electronic devices, run by electricity and influenced in their failings by electrical and physical forces. Barring removing each memory and running sophisticated electrical tests on it, then, there is only one serious memory test option: test each bit in combination with each and every other bit in the device. Unfortunately, that is impractical, since there are 393,216 bits in a 48K system, and testing every one would result in over 150,000,000,000 separate tests – a task that would take nearly a solid month running at the TRS-80's 1.77 MHz clock rate!

The remaining option, then, is to make reasonable electrical tests that exercise the neighboring bits in a memory chip, and read and write to them about as fast as the TRS-80 is likely to do it in real time. The process I have chosen is twentyfold:

1. Write a value to memory.
2. Read the stored value and check its accuracy.
3. Change the surrounding bits from zeros to ones.
4. Change the surrounding bits from ones to zeros.
5. Read the stored value and check its accuracy.
6. Change the surrounding bits from zeros to ones.
7. Change the surrounding bits from ones to zeros.
8. Write the value to memory again.
9. Read the stored value and check its accuracy.

10. Change the surrounding bits from zeros to ones.

11. Read the stored value and check its accuracy.

12. Change the surrounding bits from ones to zeros.

13. Read the stored value and check its accuracy.

14. Change the surrounding bits from zeros to ones.

15. Write the value to memory again.

16. Read the stored value and check its accuracy.

17. Change the surrounding bits from ones to zeros.

18. Write the value to memory again.

19. Read the stored value and check its accuracy.

20. Increment the value to be written and repeat.

This process is repeated 256 times, writing values from 00 to FF, and producing the electrical switches noted above. The program then moves itself to another area of memory, and checks the area in which it was just residing.

The entire memory test is still very time-consuming, since the address under test is displayed while the process is continuing.

In the program presented in Listing 3-(?), the test displays the memory location under test; any failed memory location is displayed on the screen, along with the bits which have failed. The test is presented in a 16K version with changes for 32K and 48K systems.

## Cassette I/O

One of the most maligned aspects of the TRS-80 is its cassette loading procedure. Interestingly, it is a lengthy and skillfully designed piece of coding, a victim of a combination of poor hardware (an inexpensive cassette recorder), the inclination personal computer owners have to purchase the least expensive tapes they can find, and the lack of foresight on the part of the engineers designing the routines. But there's no question that with a good tape recorder and reasonable tape, it works well. Here's how.

The routine to read and accept serial information is fairly convoluted, collapsed to about a dozen major subroutine CALLs. We will start with the SYSTEM command; since BASIC programs have other bytes to juggle (looking for out-of-memory errors, etc.), we won't tackle its major routines.

### The SYSTEM module

The SYSTEM command is evaluated by the BASIC interpreter, and its control routine is entered at 02B2. If you don't want to know how this command gets to work, then skip right to the tape loading routine two paragraphs below. An initial CALL is executed to DOS link 41E2, which in Level II merely executes a RETurn. The stack is set up at 4288, and another CALL executed to 20FE, which checks the DOS link at 41C1, picks up the 'device type' – video, tape, or printer – (video at this time), displays a carriage return, checks and saves port FF status (32 or 64 character mode and cassette state), clears the accumulator, and returns. This is preparatory housekeeping.

The accumulator is set up with a star, it is displayed (with more housekeeping), and the INPUT routine is CALLED from location 1BB3. This is the same routine used for INPUT statements, and it displays a question mark, evaluates the input line, discards everything after certain punctuation, and returns the evaluated line to the CALLing program. If a BREAK is discovered, the program returns to READY. Spaces, line feeds, tabs, etc., are cleaned out, and a syntax error is declared if no alphanumeric characters are found. If a slash (/) is found, the SYSTEM program jumps past its loading routines, picks up the start address from 40DF (more about that later), cleans out blanks

again, and evaluates the string after the slash as an interger (a CALL to 1E5A). The whole business starts over if a non-numeric string is found. If, at last, the program does discover that a number was input, the SYSTEM module is executed from the starting address stored at 40DF.

### Build-a-Byte

The first major loading call is to 0293, which searches for a synchronization byte. Since this will eventually call the 'build-a-byte' routine, let's move there first. It begins at 0241; BC and AF registers are saved. Then:

0243	DB FF	IN	A, (FF)
0245	17	RLA	
0246	30 FB	JR	NC, 0243
0246	06 41	LD	B, 41
024A	10 FE	DJNZ	024A

Port FF is checked repeatedly by inputting the value to the accumulator and rotating that value into the carry flag. If no carry is found – i.e., no 'one' bit has yet triggered port FF – the program loops back to 0243. Once a bit is found, the B register is loading with 41, and a 'waste time' loop is executed at 024A (a total of just under 500 microseconds). A CALL is then executed to 021E. Let's have a look at that:

021E	21 00 FF	LD	HL, FF00
0221	3A 30 40	LD	A, (4030)
0224	A4	AND	H
0225	B5	OR	L
0226	03 FF	OUT	(FF), A
0228	32 30 40	LD	(4030), A
022B	C9	RET	

This curious subroutine seems to stumble through checking port FF for its video state, then resetting the OUTSIG flip-flop (see the *Technical Reference Handbook* for details on this circuitry). Isn't a byte ANDed with FF and ORed with 00 merely itself? True enough, but since this is also called as a subroutine entering at 0221, with a different value for HL, the complex AND/OR strategy makes sense.

So at this point we have picked up a bit from tape, delayed, and reset the flip-flop, readying it for the next bit to trigger it. Another delay loop follows (over 850 microseconds), and a byte is input to A from port FF:

0253	DB FF	IN	A, (FF)
0255	47	LD	B, A
0256	F1	POP	AF
0257	C8 10	RL	B
0259	17	RLA	
025A	F5	PUSH	AF

The input byte is saved in the B register, and the previously saved value of A is restored from the stack. Here is a wonderful piece of serial-to-parallel conversion – a sort of software shift register. Bit 7 of port FF was input to A and saved in B, and is then rotated left into the carry flag. Then the accumulator is rotated left, bringing the state of the carry flag into bit 0 of A. The accumulator is then saved once more on the stack. Another CALL to 021E resets the port FF flip-flop, both registers are restored, and the subroutine returns to the calling program.

You'll notice that at this point we have only one bit saved in the accumulator. An eight-iteration loop would be necessary to create a whole byte . . . and it will be done. But for the moment let's see how this routine is used in the initial syncing program, which we were about to enter at 0293.

The routine's first action is to CALL 01FE. This is a detailed routine to determine the drive number and other parts of the syntax, the state of port FF (again), select the drive and get it moving. Examining the code will show that it also uses the routine entered at 0221, but with a value of FF04 in HL; this routine won't be covered here, but it is worth looking at.

The find-sync-byte routine thus turns on the tape, saves the HL register, clears the accumulator, and calls the 'build-a-byte' routine at 0241. Since this is the synchronization process, no loop value is specified:

0297	AF	XOR	A
0298	CO 41 02	CALL	0241
0299	FE A5	CP	A5
0290	20 F8	JR	NZ,0298

It continually seeks bits, endlessly rotating the accumulator until it assembles a serial stream which matches A5 (i.e., binary 10100101 – nice and symmetrical). This routine is so accurate, in fact, that whenever tape motor start-up is not a consideration, the leader consisting of zero bytes would be unnecessary. The leading '1' of A5 serves as a kind of serial 'start bit' – and the routine at 0241 handles it from there.

Any kind of match to sync byte A5 might be found, tough, since the serial stream coming in from the tape does not distinguish start and end of byte. For example, the byte pattern DD 28 also contains an A5 embedded in it. As a serial stream, DD 28 is

```
1101110100101000
....10100101....
```

- with the A5 appearing at the junction of DD and 28. So once the matching A5 is found, a return is executed to the main SYSTEM loading module. That module then CALLs a subroutine at 0235, which is a gussied-up bit reader. BC and HL are saved, then:

0237	06 08	LD	B,08
0238	CO 41 02	CALL	0241
023C	10 F8	DJNZ	0239

There's the byte read . . . read a bit with eight iterations. HL and BC registers are restored, and the subroutine returns to the main program.

## Loading the Code

The SYSTEM module now compares the byte it created with the value 55, the code assigned to machine language programs. It loops until it finds that code, then proceeds:

02D8	06 06	LD	B,6
02DA	7E	LD	A,(HL)
02D8	97	OR	A
02DC	28 09	JR	Z,02E8
02DE	CO 35 02	CALL	0235
02E1	9E	CP	(HL)
02E2	20 E0	JR	NZ,02C1

Above, the B register is loaded with the number of characters to be found in the SYSTEM program's name. The accumulator is set up with the first character of the name as entered on the \*? command line. The accumulator is tested for zero, and skips out of the loop when the end of the entered name is found. Each character following the name is read into the accumulator (CALL 0235) and compared with each letter of the entered name. If at any point the entered name does not match the name on tape, the program goes back to searching for 55 (machine program indicator) and the name search begins again.

There is a minor flaw in this process. Let's look at the succeeding lines of code:

02E4	23	INC	HL
02E5	10 F3	DJNZ	02DA

This coding increments the HL register to the next character and loops back, looking for a total of six letters in the name. But what if the machine program code (55) is found, and one or more characters of the name match, but the rest do not match? There is no provision in this routine to decrement the HL register pair . . . which means that, if only part of a correct name has been found, the program will begin its search anew until it finds a program that matches only the last part of the entered name! This is the

reason the SYSTEM routine is not always able to search until it finds the correct program, the way the BASIC load does.

Let's assume the best – that a machine program was found with the name as entered from the keyboard. A CALL is then made to 022C, where the star or space at 3C3F is toggled (XORed) with 0A. Star XOR 0A is a space, and space XOR 0A is a star; easily done.

The SYSTEM Module Continues —

02EA	CD 35 02	CALL	0235
02E0	FE 78	CP	78
02EF	28 B8	JR	Z,02A8
02F1	FE 3C	CP	3C
02F3	20 F5	JR	NZ,02EA

– searching for either 78 (end of program code) or 3C (beginning of data block code). If 78 is found, the program skips back to 02A9, where a CALL is executed to 0314. This subroutine merely reads the last two bytes on tape into the HL register, preparing the start address. This is saved at 40DF, the cassette recorder is turned off (CALL 01F8), and the SYSTEM module is re-entered from the start at 02B2. This module is a continuous loop, allowing a group of machine-language programs to be entered sequentially. Only the presence of the slash-start address combination will break out of the loop.

If a 3C is found, the beginning of a block of machine code is assumed. (If neither is found, the program loops until it finds one or the other). Here's a snippet of code:

02F5	CD 35 02	CALL	0235
02F8	47	LD	B,A
02F9	CD 14 03	CALL	0314
02FC	85	ADD	A,L
02FD	4F	LD	C,A

A byte is read and saved in B. At 0314, two bytes are read and saved, respectively, in the HL register pair. These three bytes are, first, the number of bytes to read, and second, the two-byte starting address of the block. The 0314 subroutine leaves the value transferred to H in the accumulator; to it is added the value in L, and this number, sans carry value, is saved in the C register. The C register will be used to calculate the checksum for the block being read.

## Curious Checksum

Each succeeding byte is read from tape and placed at the address now specified by HL. That byte is also added to the C register to update the simple checksum. HL is incremented to the next contiguous address, and the loop is iterated until B (the number of bytes to read in the block) reaches zero.

When the block is fully read, another byte is read from tape. This is the checksum byte, and should match the last updated value in the C register. If it does match, the program loops back, toggles the star, and begins anew the search for end-of-program (78) or block header (3C).

A correct checksum byte, curiously enough, is not a necessary element of the SYSTEM module. If the checksum is incorrect, the program will display a 'C' at video location 3C3E, and loop back regardless to continue reading the program from tape. I first noticed this action when a gentleman from New Hampshire called; he had been using a tape duplication routine to make a corrected copy of a machine language program. He had loaded the tape, returned to BASIC, then POKed in a few byte changes. He then continued with the duplication. When he loaded the tape later on, he got a 'C' error message on the screen... but the program continued to load and did execute properly. The checksum was wrong because of the byte changes he had made, but the program, checksum notwithstanding, was read and loaded completely.

Let's take a look at that final portion of code:

02FE	CD 35 02	CALL	0235
0301	77	LD	(HL),A
0302	23	INC	HL
0303	B1	ADD	A,C
0304	4F	LD	C,A
0305	10 F7	DJNZ	02FE
0307	CD 35 02	CALL	0235
030A	B9	CP	C
030B	28 DA	JR	Z,02E7
030D	3E 43	LD	A,43
030F	32 3E 3C	LD	(3C3E),A
0312	18 06	JR	02EA

Overall, these routines give the appearance of being reasonable and reliable, and they should be. What, then, gives rise to the tape problems? Mostly the timing loop in the 0235/0241 subroutine. The values placed in the B register at 0248 and 024F are too short for low-grade audio processing. Simply stated, the audio waveform coming in from tape 'rises' too slowly for the fast bit-check loop at 0251 to catch. A 'one' might come through, but it comes through too laggardly for port FF to have flipped into place.

## Special Loaders

This was initially one of the mysteries of TRS-80 operations. *Microchess* was produced with a loader, then others quickly followed, mysteriously taking control of the machine and locking it up completely.

Let's now take a look at some of these special loaders, which will be designated Loaders A, B, C, and D in order to help them continue to do the job they were supposed to – protect software.

Loader A sets up a stack at 5000, clears the accumulator, and calls ROM to turn on the tape recorder and find the sync byte. It places a star on the bottom of the screen, sets up the HL register to receive the program, and prepares register C to perform simple checksum. A byte is read, it is saved in memory, and the checksum is created as in the SYSTEM mode. Then:

4025	7C	LD	A,H
4026	1F	RRA	
4027	23	INC	HL
4028	3E 2A	LD	A,2A
402A	DA 2F 4D	JP	C,402F
402D	3E 20	LD	A,20
402F	32 F0 3F	LD	[3FF0],A
4032	3E 4C	LD	A,4C
4034	8C	CP	H
4035	C2 1F 4D	JP	NZ,401F
4038	3E FF	LD	A,FF
403A	80	CP	L
403B	C2 1F 4D	JP	NZ,401F
403E	89	CP	C
403F	C2 00 00	JP	NZ,0000
4042	C0 F8 01	CALL	01FB
4045	C3 80 47	JP	4780

The strange appearance of RRA has nothing to do with rotating incoming bits. Rather, since the accumulator contains the H register value, each page (256 bytes) of information will change the high page value by one. Consequently, the high page will alternate between odd and even values, and the least significant bit, rotated into the carry flag, will trigger the display-star or display-space routines at 4D2F.

Finally, this somewhat awkward loader does a pair of compares to see if it has yet reached 4CFF, the end of the program load. If not, it loops back and continues; if so, it examines the checksum in C. Amazingly enough, it goes back to MEMORY SIZE? if there is a checksum error! There's no tampering with this program. A successful load is followed by a jump to the program's beginning at 4780.

Loader B is virtually identical to Loader A, except that the beginning of the program is found at 41FD instead of 4780.

Loader C is of a more interesting variety. It is written entirely without calls to ROM, because it

```

00100 ; #####
00110 ; VOICE INPUT/OUTPUT ROUTINE USING THE CASSETTE PORT AND
00120 ; AMPLIFICATION. CAN BE USED WITH CTR TAPE RECORDERS AND
00130 ; BUILT-IN MICROPHONES OR PREFERABLY EXTERNAL CRYSTAL
00140 ; MICS. SMALL SPEAKER OUTPUT INCREASES INTELLIGIBILITY.
00150 ; #####
00160 ;
4300 00170 ORG 4300H ; LOW POINT IN MEMORY
00180 ORG 6500H ; USE WITH DISK BASIC
06CC 00190 MONITR EQU 06CCH ; BASIC EXIT (OR OTHER)
4300 F3 00200 START OI ; NO BOTHERSOME STUFF
4301 CDC901 00210 CALL 01C8H ; CLEAR THE SCREEN
4304 3A3040 00220 LD A,(4030H) ; START BY RESETTING PORT
4307 03FF 00230 OUT (0FFH),A ; TO CLEAR INCOMING BITS
00240 ;
00250 ; #####
00260 ; KEYBOARD ROUTINE FOR ENTER (INPUT), CLEAR (OUTPUT), OR
00270 ; UP-ARROW (BASIC). UP-ARROW GOES TO EXIT IF NOT BASIC.
00280 ; #####
00290 ;
4309 3A4038 00300 KEYTST LD A,(3840H) ; GET ENTER/CLEAR ROW
430C FE01 00310 CP 1 ; CHECK IF ENTER PRESSED
430E 2808 00320 JR Z,INPUT ; GO TO INPUT ROUTINE
4310 FE02 00330 CP 2 ; CHECK IF CLEAR PRESSED
4312 2850 00340 JR Z,OUTPUT ; GO TO OUTPUT ROUTINE
4314 FE08 00350 CP 8 ; CHECK FOR UP-ARROW
4316 CACC06 00360 JP Z,MONITR ; OUT TO BASIC OR MONITOR
4319 18EE 00370 JR KEYTST ; BACK FOR A VALID KEY
00380 ;
00390 ; #####
00400 ; INPUT FROM PORT FF (255 DECIMAL) AND STORAGE IN MEMORY
00410 ; #####
00420 ;
431B 21A343 00430 INPUT LD HL,MSG01 ; GET THE "INPUT" MESSAGE
431E CDA728 00440 CALL 28A7H ; AND DISPLAY ON SCREEN
4321 3A3040 00450 LD A,(4030H) ; GET VALUE FOR PORT MASK
4324 4F 00460 LD C,A ; SAVE MASK IN C REGISTER
4325 210044 00470 LD HL,4400H ; BEGIN VOICE STORAGE
00480 ; LD HL,6700H ; BEGIN STORAGE (DISK)
4328 1608 00490 LOOP1A LD 0,B ; NUMBER OF BITS IN BYTE
432A 0BFF 00500 LOOP2 IN A,(0FFH) ; GET VALUE AT THE PORT
432C CB17 00510 RL A ; STASH IT IN CARRY BIT
432E CB13 00520 RL E ; RUMP IT INTO E REGISTER
4330 79 00530 LD A,C ; GET VALUE OF PORT MASK
4331 3A4038 00540 LD A,(3840H) ; CHECK ENTER/CLEAR ROW
4334 FE80 00550 CP 80H ; CHECK IF SPACE PRESSED
4336 C25343 00560 JP NZ,ESCAPE ; OUT IF KEYBOARD CLEAR
00570 ;
00580 ; #### NOTE: DELAY VALUE USED IN THE B REGISTER IS ####
00590 ; #### CHOSEN FOR OPTIMUM INTELLIGIBILITY.WITH THE ####
00600 ; #### CTR TAPE RECORDER AND HARDWARE MODIFICATION. ####
00610 ; #### A LONGER VALUE CAN BE USED IF HIGH-FIDELITY ####
00620 ; #### INPUT IS PROVIDED. FOR EACH INCREASE IN THE ####
00630 ; #### B-REGISTER DELAY VALUE, ALSO INCREASE THE B- ####
00640 ; #### REGISTER BY THE SAME AMOUNT FOR PLAYBACK. ####
00650 ; #### LIKEWISE, A DECREASE IN THE DELAY VALUE MAY ####
00660 ; #### INCREASE FIDELITY AT A SACRIFICE OF MEMORY. ####
00670 ;
4339 0604 00680 LD B,4 ; GET SHORT DELAY VALUE
433B 10FE 00690 DELAY1 DJNZ DELAY1 ; AND DELAY A WHILE
433D 03FF 00700 OUT (0FFH),A ; MUST RESET PORT INPUT
433F 15 00710 DEC 0 ; DECREMENT TOTAL BITS
4340 C22A43 00720 JP NZ,LOOP2 ; CONTINUE IF MORE TO GO
4343 73 00730 LD (HL),E ; SAVE FULL BYTE IN MEM.
4344 23 00740 INC HL ; GO ON TO NEXT BYTE
4345 7C 00750 LD A,H ; GET VALUE OF M.S. BYTE
4346 FE00 00760 CP 00H ; USE FOR 48K MACHINE
00770 ; CP 0C0H ; USE FOR 32K MACHINE
00780 ; CP 0B0H ; USE FOR 16K MACHINE
4348 C22843 00790 JP NZ,LOOP1A ; IF NOT DONE THEN MORE
434B 21C743 00800 LD HL,MSG02 ; GET "INPUT COMPLETE"
434E CDA728 00810 CALL 28A7H ; AND DISPLAY THE MESSAGE
4351 18B6 00820 JR KEYTST ; DONE - BACK TO KEY TEST
00830 ;
00840 ; #####
00850 ; PAUSE CHECK DURING ENTRY; SPACEBAR = GO, OTHERWISE STOP
00860 ; #####
00870 ;
4353 E5 00880 ESCAPE PUSH HL ; SAVE CURRENT POINTER
4354 CDAF0F 00890 CALL 0FAFH ; DISPLAY CURRENT MEM.
4357 213944 00900 LD HL,MSG05 ; GET "WORD START" MESS.
435A CDA728 00910 CALL 28A7H ; AND DISPLAY THE MESSAGE
435D E1 00920 POP HL ; RESTORE MEMORY PTR.
435E 3A4038 00930 RECHK LD A,(3840H) ; ENTER/CLEAR KEYBRO ROW
4361 FE80 00940 CP 80H ; CHECK IF SPACE AGAIN
4363 28C3 00950 JR Z,LOOP1A ; BACK TO MAIN LOOP
4365 FE04 00960 CP 4 ; CHECK IF BREAK KEY
4367 20F5 00970 JR NZ,RECHK ; KEEP LOOKING ENT OR BRK
436B 21C743 00980 LD HL,MSG02 ; GET "INPUT COMPLETE"
436C CDA728 00990 CALL 28A7H ; AND DISPLAY THE MESSAGE
436F 18B6 01000 JR KEYTST ; AND BACK TO KEY MENU
01010 ;

```

Listing Continued . . .

## Continued Listing

```

01020 ; #####
01030 ; OUTPUT FROM MEMORY OF RECORDED VOICE TO CASSETTE PORT
01040 ; #####
01050 ;
4371 21E743 01060 OUTPUT LD HL,MSG03 ; GET "BEGIN OUTPUT"
4374 CDA728 01070 CALL 28A7H ; AND DISPLAY THE MESSAGE
4377 3A3040 01080 LD A,(403DH) ; PORT FF OUTPUT MASK
437A 4F 01090 LD C,A ; SAVE OUTPUT MASK IN C
437B 210044 01100 LD HL,4400H ; START VOICE STORAGE (*)
437E 160B 01110 LOOP3A LD D,8 ; NUMBER OF BITS IN BYTE
4380 7E 01120 LD A,(HL) ; GET VALUE FROM MEMORY
4381 5F 01130 LD E,A ; SAVE IT IN E REGISTER
4382 AF 01140 XOR A ; CLEAR ACCUMULATOR TO 0
4383 CB13 01150 LOOP4 RL E ; SEND BIT TO CARRY FLAG
4385 CB17 01160 RL A ; AND ROTATE 'ROUND TO A
4387 81 01170 OR C ; USE THE PORT FF MASK
4388 D3FF 01180 OUT (OFFH),A ; AND SEND OUT THE VALUE
01190 ;
01200 ; ##### NOTE: PLAYBACK VALUE BELOW MUST BE CHANGED #####
01210 ; ##### TO MATCH SAMPLING DELAY IN THE INPUT SECTION #####
01220 ; ##### OF THIS I/O PROGRAM. THIS VALUE IS ROUGHLY #####
01230 ; ##### TWO TIMES THAT IN THE 8-REGISTER DURING THE #####
01240 ; ##### INPUT SAMPLING. VARIOUS DUMMY OPCODES MAY #####
01250 ; ##### BE INSERTED WHERE NECESSARY TO KEEP VOICE #####
01260 ; ##### AT THE PROPER PITCH AND QUALITY. USING THIS #####
01270 ; ##### PROGRAM, THERE IS A QUARTER-TONE DIFFERENCE. #####
01280 ;
438A 0606 01290 LD B,6 ; GET SHORT DELAY VALUE
438C 10FE 01300 DELAY DJNZ DELAY ; AND DELAY SHORT WHILE
438E AF 01310 XOR A ; CLEAR ACCUM. BACK TO 0
438F 15 01320 DEC O ; BITS = BITS MINUS ONE
4390 C28343 01330 JP NZ,LOOP4 ; AND BACK FOR SOME MORE
4393 23 01340 INC HL ; GET NEXT BYTE FROM MEM.
4394 7C 01350 LD A,H ; GET VALUE OF M.S. BYTE
4395 FE00 01360 CP 00H ; FOR 48K MACHINE
01370 ; CP 0C0H ; FOR 32K MACHINE
01380 ; CP 080H ; FOR 16K MACHINE
4397 C27E43 01380 JP NZ,LOOP3A ; AND GO BACK FOR MORE
439A 210E44 01400 LD HL,MSG04 ; GET "OUTPUT COMPLETE"
439D CDA728 01410 CALL 28A7H ; AND DISPLAY THE MESSAGE
43A0 C30943 01420 JP KEYTST ; AND BACK WHEN DONE
01430 ;
43A3 48 01440 MSG01 0EFH 'HOLD SPACE BAR AND BEGIN SPEAKING.'
43C5 00 01450 0EF8 00H
43C6 00 01460 0EF8 00H
43C7 49 01470 MSG02 0EFH 'INPUT COMPLETE OR MEMORY FULL.'
43E5 00 01480 0EF8 00H
43E6 00 01490 0EF8 00H
43E7 42 01500 MSG03 0EFH 'BEGINNING PLAYBACK; BREAK IS IGNORED.'
440C 00 01510 0EF8 00H
440D 00 01520 0EF8 00H
440E 50 01530 MSG04 0EFH 'PLAYBACK COMPLETE; PRESS CLEAR TO REPEAT.'
4437 00 01540 0EF8 00H
443B 00 01550 0EF8 00H
4439 20 01560 MSG05 0EFH ' = WORD SEPARATION POINT.'
4452 00 01570 0EF8 00H
4453 00 01580 0EF8 00H
01590 ;
01600 ; #####
01610 END START
4300
00000 TOTAL ERRORS
29121 TEXT AREA BYTES LEFT

DELAY 438C 01300 01300
DELAY1 439B 00690 00690
ESCAPE 4353 008B0 00560
INPUT 431B 00430 00320
KEYTST 4309 00300 00370 00820 01000 01420
LOOP1A 4328 00490 00790 00950
LOOP2 432A 00500 00720
LOOP3A 437E 01110 01390
LOOP4 4383 01150 01330
MONITR 06CC 00190 00360
MSG01 43A3 01440 00430
MSG02 43C7 01470 00800 00980
MSG03 43E7 01500 01060
MSG04 440E 01530 01400
MSG05 4439 01560 00900
OUTPUT 4371 01060 00340
RECHK 435E 00930 00970
START 4300 00200 01610

```

is capable of loading into a Level I or Level II TRS-80. Less fortunately, the ROM timing errors are not corrected, so the chances of loading this program on a marginal machine are not at all improved. The stack is prepared, and a block of memory is cleared from 5800 to the end of potential RAM at FFFF. My only guess as to the reason for this is that the authors wish to wipe out any programs such as monitors or disassemblers, as the clearing byte (A5) does not strike me as otherwise meaningful.

The tape is then turned on, and a pattern of three asymmetrical and two symmetrical sync bytes is found (B1, 83, 79, 5A, 00). Again, the choice strikes me as arbitrary, and may be the authors' way of identifying their own code. If these bytes are found, the program continues; if not, the entire five-byte pattern is sought again.

As in the other loaders, register C is set to zero for use as a checksum byte. The program load point is set high in memory (747F), and a byte is read. Here is a part of the code:

```

4330 C0 BF 43 CALL 43BF
4340 77 LD (HL),A
4341 32 3F 3C LD (3C3F),A
4344 B1 ADD A,C
4345 4F LD C,A
4346 2B DEC HL
4347 70 LD A,L
4348 3C INC A
4349 C2 53 43 JP NZ,4353
434C C0 BF 43 CALL 43BF
434F 8B CP C
4350 C2 66 43 JP NZ,4366

```

The secret to this portion of code rests in address 4346. Unlike most other loaders, this one loads (and displays) the last byte of code first, moving backwards through memory. (438F is the location of the byte-read subroutine). When the page is crossed (4346-4348), the checksum is evaluated; if the checksum is incorrect the program jumps to 4366, where an error message is displayed and the machine locks up.

The user's display is worth noting:

```

4353 7C LD A,H
4354 32 3E 3C LD (3C3E),A

```

This loader actually displays the ASCII equivalent of the page of memory being loaded with data . . . and it looks like an alphanumeric countdown as the program is fit into place.

Finally, Loader C does a comparison for the end of the first major load block, changes the value of H, and loads the next block. It then overwrites critical portions of the load routine, effectively obscuring the loading and entry point of the program. Interrupts are disabled, and the

## Conclusions

process moves out of the loader into the main program. Interestingly, the authors forgot to turn the tape recorder off.

Finally, Loader D is of an entirely different sort. First, some code:

BEFE	3E D4	LD	A,4
BF00	03 FF	OUT	(FF),A
BF02	0B FF	IN	A,(FF)
BF04	17	RLA	
BF05	3D FB	JR	NC,BF02
BF07	06 XX	LD	B,XX
BF09	1D FE	DJNZ	BF08
BF0B	06 0B	LD	B,9
BF0D	3E D4	LD	A,4
BF0F	03 FF	OUT	(FF),A
BF11	0B FF	IN	A,(FF)
BF13	17	RLA	
BF14	00	NOP	
BF15	3B DC	JR	C,BF23
BF17	23	INC	HL
BF1B	2B	DEC	HL
BF1B	1D F6	DJNZ	BF11

This remarkable loader is written for high-speed operation, setting up the output ports

BEFE and BF0D), clocking itself with start bits (BF0D), and then reading a nine-bit serial stream. Careful timing and self-clocking are essential in high-speed data I/O, and this routine is capable of reading and writing on ordinary audio cassettes, with excellent reliability, at better than 2000 baud. The only point to the instructions at BF17 and BF18, for example, is the delay introduced by executing them; yet that timing is very important. The actual timing value at BF07 has been dropped for a measure of protection of this author's fine software.

## Conclusions

In sum, the tape read/write routines of the TRS-80 are efficient and, especially now with special loaders and a corrected ROM, quite reliable. Different levels of user prompts, particularly those used by the reverse-loading module described above, are probably more satisfactory than flashing stars. A checksum process for BASIC similar to the SYSTEM module would have been valuable. Finally, by careful attention to clocking details, a reliable, higher speed loader could have been included in the TRS-80.

For those especially interested in high-speed loaders, I recommend examining the *Exatron Stringy-Floppy* operating system, which shows what can be done with equipment designed for digital operation. It is capable of reliable loading and saving at rates exceeding 11,000 baud.

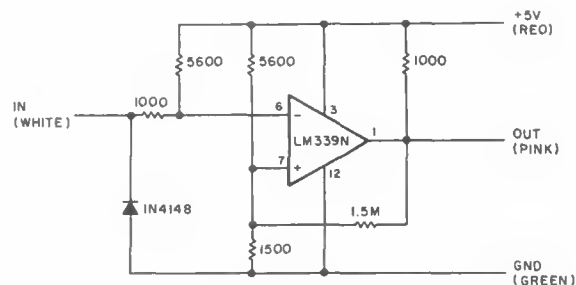


Fig. 2. Full schematic for the Model I cassette modification for speech input. It should be switched out when cassette programs are being loaded (see Fig. 4).

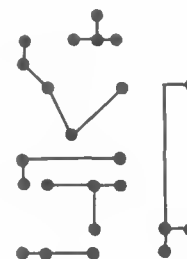


Fig. 3a.

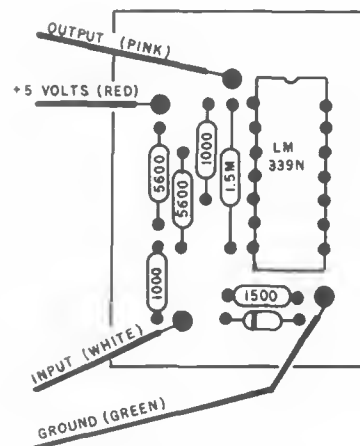


Fig. 3b.

# NOTES

---

# 4

---

## Simple Modifications

Why hardware modifications? Simply because they help a limited computer broaden its ability to serve our needs. Looking at glowing blue-white letters on a dark screen is comfortable for only a limited time . . . waiting through many minutes for the computer to make a game play is frustrating . . . attempting to imagine which characters are upper case and which are lower case is nearly impossible . . . emphasizing individual words and characters is only possible with arrows and stars . . . and so forth.

In this chapter, eleven simple modifications to the standard TRS-80 will be presented:

Installing 16K memory to the keyboard unit, and to the expansion interface.

A change to enable recovering Resets when there is an expansion interface attached.

Moving the Reset button to a more accessible position.

Adding both an extra keyboard and an extra video output for extending the computer's portability.

Adding an RF modulator to further increase the TRS-80's portability.

Installing a modification to increase the computer's computational speed.

Installing a modification to reverse the video display for black letters on a white background, more in keeping with real text.

Adding the feature of dual-language operation: Level I and Level II in the same computer.

Accessing the lower case capabilities built into the TRS-80 by adding a single circuit.

Reversing individual characters on the screen for emphasis.

Adding a hexadecimal keypad for faster entry of machine code information.

Almost all of these modifications will require you to open the cabinet of your TRS-80. Should you do it? Or should you not? First of all, consider that your computer has but a three month warranty; after that, you will have to pay repair charges should something go wrong. Radio Shack has, in response to public protests, changed its former policy and will now repair machines with the most popular modifications installed, at no extra charge. They will not remove those modifications if you say so, but may refuse repair if (a) you have mangled the board, or (b) if you cannot document your modifications.

Yes, there is a chance that you can damage your TRS-80 when you make modifications. This will not be the fault of the modifications themselves, though. My own TRS-80 has been modified many times, and the only damage has been to the notoriously flakey cable that connects the keyboard to the main circuit card, and the failure of one RAM chip. The latter would have happened anyway.

## Keep It Clean!

How many TRS-80's have been damaged during modification? That's a hard question to answer, but I will describe briefly the failed 80's I have seen:

1. A damaged integrated circuit in the data section was caused when heavy wire was used to make the modification.

**Lesson** — use the parts specified.

2. Two RAM chips were blown when the user dropped a soldering iron into the unit when he was modifying it.

**Lesson** — always turn the power off when making the physical changes.

3. A blown power supply regulator. The user had left the unit under an open window.

**Lesson** — close the windows in case of rain!

4. High levels of garbage causing keyboard lockout and blown programs. The computers were being used next to heavy electrical equipment.

**Lesson** — treat the computer as if it were a very sensitive piece of electronic equipment. It is.

5. Constant keyboard lockout or odd characters. The TRS-80 cable had been flexed too many times, causing breaks. These breaks had destabilized the keyboard circuits, causing two ICs to blow.

**Lesson** — handle the keyboard cable, that most delicate of hardware, with utmost care.

6. Constant lost memory and programs. The RAM chips were not being 'refreshed' because the refresh multiplex line was not working. The user had plugged in a peripheral device upside down, with the power on to both.

**Lesson** — make all interconnections with the power off.

In sum, all the above damage was caused by carelessness, haste, or attempting to use inappropriate materials to do the job. The solutions? Resolve not to do all the work in one evening. Turn back to the introduction to make sure you have the right tools. Work slowly and take breaks often. Buy parts from reliable suppliers. Read and understand the instructions *and the theory* before you start.

And finally, if you have any serious doubts about the accuracy of the printed information, contact the author. That's me. And if you have any problems that won't go away, write. If you write, you *must* enclose a self-addressed, stamped envelope, a complete description of any problem, and all tests you have made. Also, I cannot cover much beyond the scope of this book, which includes the myriad competing disk operating systems and support software.

## Expanding the Memory

The simplest modification, once you have braved opening the cabinet, is expanding the unit's memory from the 4K supplied to a full 16K RAM. At this writing, 16K of reliable RAM memory can be purchased for less than \$20.

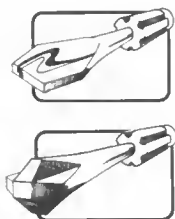
## Keep It Clean!

If you examine the contacts on the keyboard unit's edge-card connector, as well as on the five connectors (expansion in, expansion out, printer, RS-232, disk) of the Expansion Interface, you will see a major cause of the TRS-80's instability: solder-plated connectors. In the interest of economy, Radio Shack did not use gold to ensure a good cable contact. This turned out to be a serious error in judgment.

There are two options in dealing with these solder-plated connectors. The first is to keep them clean. Remove the cables regularly and vigorously rub the contact surfaces with a dollar bill or talc buffing wheel. Bring the

solder to a bright shine, and spray it with contact cleaner. Reinstall the cables and memory crashes should lessen.

The other option is expensive and time-consuming, but much more reliable. The contacts can be freed of solder plating with solder wick, cleaned with flux remover, brushed to a high shine, and a soft silver compound can be flowed onto the edge contacts. Fuller Software (see Appendix I) sells a contact plating kit; price varies with the price of silver, but at this writing it is more than \$20. The process is very tedious because silver melts at a much higher temperature than lead, but the results are noise-free connections and greater reliability.



There are several types of so-called '16K memories', so when you set out to upgrade your TRS-80 memory, make sure you order the correct type. 16K is actually a shorthand term for 16,384 bytes of memory. In the TRS-80, the 16K memories are integrated circuits containing 16,384 single-bit memory cells each. To create an entire byte, then, eight integrated circuits are needed. Furthermore, the TRS-80 memory needs very little power, and must take only a small amount of space in its cabinet. The only small, low-power memories made are 'dynamic' memories. The 16K-by-one-bit, dynamic memory is industry type 4116, also called type 416.

There is one other consideration in purchasing memory expansion chips, and that is a popularly misunderstood quality called *access time*. Access time can be thought of as the time it takes the computer to inform the memory chip that it needs information (or will give it information) until the memory chip is electronically ready to respond. This figure is usually given in *nanoseconds* (see Table 4-1).

Second	1
Millisecond (mS)	0.001
Microsecond (uS)	0.000001
Nanosecond (nS)	0.000000001

Table 4-1. Relative time units.

This time is very small but quite critical to the operation of the computer. If the memory is not ready, programs and data will be recorded or reported incorrectly, and the computer will have no chance of working properly. The minimum access time for memory in an unmodified TRS-80 is 450nS; with the speed-up modifications presented in this book, that figure drops to 300 nS or less.

Most of the current crop of memory chips easily meet the 450 nS requirement, and many of those sold as 450 nS chips can be run reliably at 300 nS. A problem arises with the expansion interface, as older units had a bit of trouble with 'hotter' (faster) memory. So as a general rule, refer to Table 4-2 when looking for memory to upgrade your TRS-80 or its expansion interface.

Equipment	Access Time
TRS-80 keyboard unit, unmodified	450 nS mex, 350 nS optimum
TRS-80 keyboard unit with speed-up	300 nS mex, 250 nS optimum
Expansion Interface made before 1/80	300 nS min, 450 nS mex
Expansion Interface made after 1/80	450 nS mex, 350 nS optimum
Expansion Interface made before 1/80 with speed-up	300 nS min optimum
Expansion Interface made after 1/80 with speed-up	300 nS mex, 250 nS optimum

Table 4-2. Memory access time for various TRS-80s.

## Opening and Closing the Case.

1. Locate a spacious workplace, and set a soft towel on it. You will need a Phillips screwdriver and a small box in which to set the screws and spacers.
2. Remove the power and disconnect all cables to the keyboard unit. Set it face down on the towel.



Photo 4-1. Photos showing opening computer.

Unit is placed face down and screws are removed. On newer units one screw is beneath a warranty notice.

3. There are six screws of three different sizes which hold the case together. In later TRS-80's, one of these is covered by a label warning you not to go inside. This label just won't peel off for later use. You'll have to punch through it to remove the last screw.
4. Hold the case together and place it face up on the towel. Gently lift off the top cover. Some TRS-80's have a 'flying lead' LED (light-emitting diode) power indicator, meaning you will have to pull it gently out of the hole in the top cover. Other power indicators are fastened to the keyboard, and the cover lifts off directly.



Computer is placed face up and the top removed by lifting back and up.

## Opening and Closing the Case

5. This step is the most delicate. Lift the keyboard slightly upward, and then swing it toward you. A cable attaches from the keyboard to the main circuit card at the bottom left side. This cable is made up of flat copper bands which have a tendency to break when flexed.



*Keyboard is rocked gently forward to show white plastic spacers, which are then removed.*

6. While holding the keyboard up at about a 90-degree angle, look at the main circuit card. There are five soft plastic white spacers which cushion the keyboard. On later models, a sixth, central spacer is hard plastic. Note their positions, remove them, and set them in the parts box.

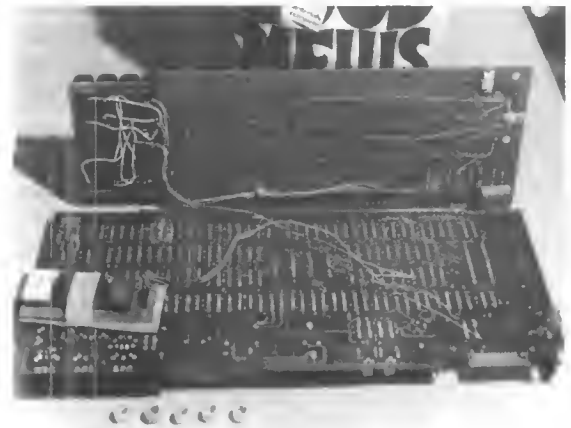
7. Swing the keyboard downward gently, and lift the entire computer out of the base of the cabinet. Set the computer down and put the base aside.



*With the keyboard supported upright against a firm backing, the CPU board is lifted from the case bottom.*

8. Most modifications will be done to the integrated circuit side of the circuit board, so turn the entire unit face down again. Unless the modification calls for work on the solder side of the board, do not flex the keyboard cable again.

9. If the modification calls for work on the bottom of the board, get a heavy box or other support. Set the keyboard face up, and swing the keyboard out and up a bit wider than 90 degrees, leaning it against the support. Never open the unit fully like a book, as this badly deforms the interconnect cable.



*The complete unit is ready for work with the keyboard supported at a 90-degree angle.*

10. To close the case, fold the unit back together, and set it into the base. The unit may have to be jostled gently to get it to fit over the plastic support posts. Make sure any added wires do not get caught and cut by the support posts.

11. Lift the keyboard slightly and restore the white spacers to their former positions. Fit the keyboard back into place.

12. If any long wires or cables have been added, make sure they all clear the cabinet edges. Restore the LED to its place on the keyboard top if it is a flying-lead type, or straighten the LED on the circuit card so it fits into the cover hole.

13. Fit the cover into place lightly, making sure there are no newly installed parts being crushed or bent in the process. Be sure no leads creep out the joints on either side.

14. Holding the unit together firmly, flip it on its face. While holding it with one hand, drop the screws into the holes, longest one towards

the top. Fasten one screw on either side; this will hold the computer together while you tighten the remainder.

15. Remember that the case is soft plastic, so just tighten enough to pull the two sides together. Otherwise, the plastic may be stripped and the screws will fall out. If that happens, drop in some clear acrylic cement (not white glue), and insert the screw. Remove it just before the glue hardens, and replace it about an hour later.

16. Flip the computer on its back, restore cables, and turn on the power. If the slightest problem seems apparent, open up and try again!

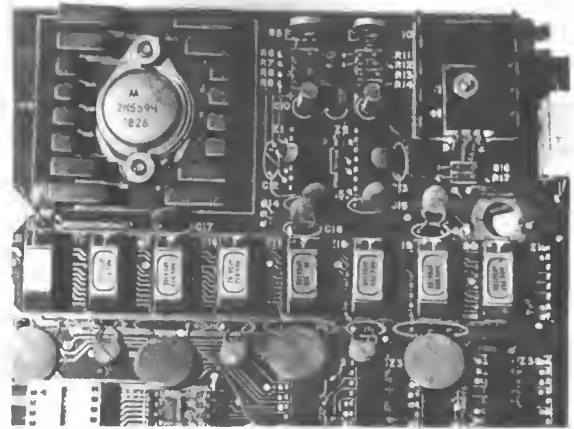


Photo 4-2. Memory chip area in TRS-80.

*Power Supply and Memory: 2N5594 transistor handles 5-volt supply; adjustments are seen at top of photo. 16K RAM chips are plugged into sockets Z13 to Z20, unusually close to the power transistor's heat sink. Replacement memory chip in socket Z13 attests to the degrading capacities of excess heat.*

## Tips on Handling Integrated Circuits

In the early days of microcomputers, there was a lot of user hesitation about installing memory chips because of warnings about static electricity damaging the memory devices. At that time the fear was reasonable; but today (with just a little caution) there need be no problem.

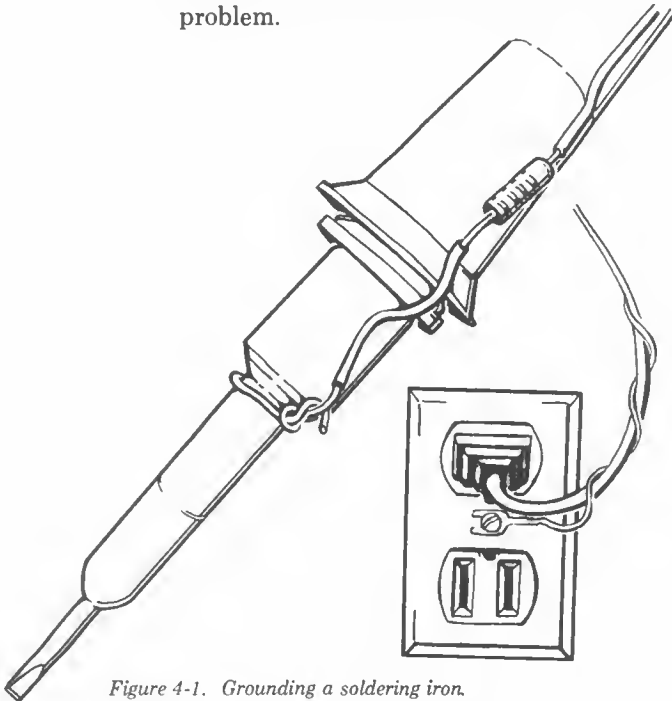


Figure 4-1. Grounding a soldering iron.

1. Never place any integrated circuit on highly charged plastic material, especially styrofoam.

2. Handle memory chips, CPUs (such as the Z-80), LSI devices (large-scale integrated circuits, usually those with 28 or 40 pins), or any marked MOS, CMOS, or NMOS (metal-oxide semiconductors), with care. Hold them by their ends, never by the connection pins.

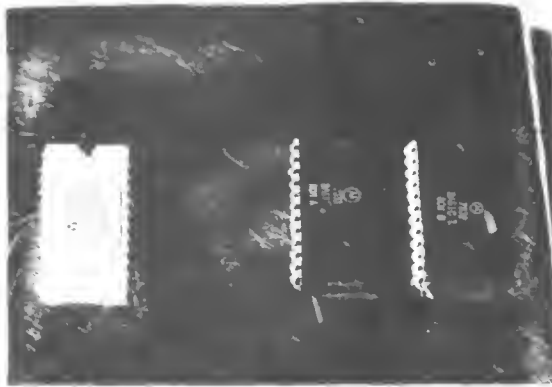
3. Purchase a static-free workbench, which is a conductive cloth sheet with a wrist strap and safe grounding cable. These can be obtained from Wescorp for about \$18.

4. Ground your soldering iron to an earth ground *but only through a series-connected one-megohm resistor — never directly!* The grounding is not absolutely essential, but helps if you live in a very dry, static-producing environment.

5. Work with any integrated circuits with the power off. Make sure the integrated circuit's ground and power pins are all connected (soldered or in sockets) before turning on the juice! A difference of a mere half a volt between certain pins can kill an IC.

6. Use high-quality sockets for integrated circuits wherever you can. This will not only keep excessive heat away from them, but will also save the day if one is damaged. Unsoldering a 40-pin integrated circuit is not pleasant.

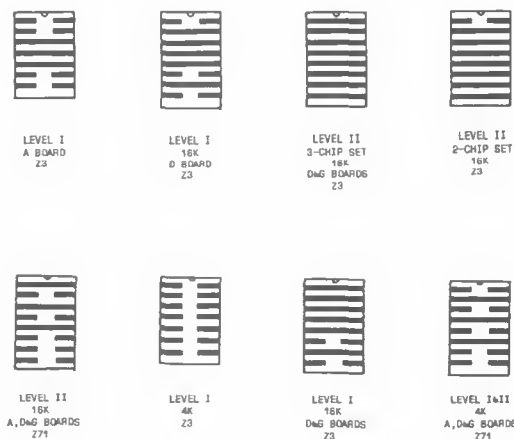
7. Above all, work slowly and carefully. By far the greatest villain is haste. Oh yes — do keep furry animals out of the area!



*Level I ROMs: Rockwell single-chip ROM and Motorola 2-chip set are pushed into aluminum-foil-covered vegetable tray.*

Before installing your new memory chips, take a styrofoam meat or vegetable tray, trim off the curved ends, and cover the center with aluminum foil. This will be your static-free storage for the 4K memory chips you will be removing. To install the 16K memory in your keyboard unit, turn off the power, open the case and find the 4K memory chips.

Slide a thin-bladed screwdriver under the end of one of these chips, and rock it slightly upwards. Slide the blade under the other side, and rock. Move back and forth gently until the chip is free, but don't spring it out of the socket. Lift it by the ends and press it into the foil-covered tray. As you are doing this, notice that each of the chips has a notch or dot at one end. Keep this position in mind; the 16K chips will be installed in the same direction.



*Figure 4-2. 16K RAM expansion shunt versions.*

Once all the 4K chips have been removed, lift the 16K chips one at a time and press them in the empty sockets. If the pins are spread too wide, set the chip sideways on the foil-covered tray and press gently. Try again to insert the chips. Be sure none of the pins are bent underneath, or slide outside the socket.

Now turn your attention to a pair of on-board sockets marked Z3 and Z72. 16K chips will need more attention from the address lines, so these decoding shunts will have to be changed. If you didn't receive these shunts with your order of 16K chips, have no fear — just slide wires into the sockets. Or, obtain an 8-position DIP switch and push it into the socket. Turn the switches *on* where the shunt is to be connected, *off* where the bars are broken. Figure 4-1 shows the old and new positions of the shorting bars or wires of these shunts.

When you have placed the new, corrected shunts, switches or wires into sockets Z3 or Z72, and all the 16K chips are installed, the keyboard upgrade is complete. Reinstall the computer in its case, turn on the power, and press (ENTER) in response to MEMORY SIZE? Then PRINT MEM, and you should see a value of 15572 (15570 on newer machines). If you don't get that value, turn the machine off and troubleshoot:

1. MEMORY SIZE? reads much less than 15570. Turn off the computer and try again.

If you have no expansion box, type SYSTEM (ENTER) /0 (ENTER), and keep trying. If the value never changes, you may have either a bad memory chip or incorrect shunt wiring. Go back to MEMORY SIZE?, only this time enter 15560. If you get the READY message, this might point to a failed memory chip or an incorrect DIP shunt (especially if PRINT MEM? reads the same as that for 4K). To test memory, run the RAM test printed in Chapter 3.

If instead you get the flash of an ?OM ERROR and a return to MEMORY SIZE?, then suspect that you've wired the shunt or shorting wires incorrectly.

2. You get a partial RADIO SHACK LEVEL II BASIC (or R/S LEVEL 2 BASIC) message, with or without READY, and with or without incorrect characters, but it only lasts for a short time before crashing back to MEMORY SIZE?

Suspect that a memory chip is very balky, is inserted only partly or with pins bent, or that you've lifted the Level II interconnect cable (if your unit has one). You may also have damaged some other circuitry, but this is very unlikely.

3. The screen never gets past a pile of garbage.

You may have lifted the Level II interconnect cable (if your unit has one), one or more memory chips may be completely dead, inserted backwards or only partly, or you may have forgotten to reinstall (or have reversed) either of the two shunts at Z3 and Z72.

4. Unexpected characters are displayed after MEMORY SIZE?, sometimes acting as if they were 'entering' themselves.

You have broken one or more wires of the keyboard interconnect cable. You can look for cracks, or just replace the whole cable.

5. The machine responds correctly, but only for a short while; it often crashes; occasionally PRINT MEM will give a smaller number than 15570, but not always.

This is probably balky memory or memory that is the wrong speed (usually older, slower memories that some discount houses may sell). For starters try reinserting the memory in case of a bad contact; run the RAM test; or just buy new memory.

6. The computer displays a screen full of 999, etc.

You have lifted the Level II interconnect cable out of its socket. Replace it very carefully.

Adding to the Expansion Interface is a much easier task. Your keyboard unit must have 16K in it already in order that the memory map be complete from 4000 (decimal 16384) to the start of expansion box memory at 8000 (decimal 32768). And, sadly, you cannot use your 4K chips in the expansion box without a hardware modification.

First, remove the cover over the power supplies and remove them; this will prevent them from tumbling all about when you open the bottom cover. Now flip the expansion box over and remove and set aside the six screws that fasten the cover. Also disconnect the power cable inside the expansion interface case.

Inside, you will find two rows of empty sockets for memory expansion. The first 16K of expansion memory goes in the sockets marked Z9 to Z16, and the second 16K into sockets Z1 to Z8. The memory must be inserted in this order, unless you want a permanently protected, 16K, high-memory block (which might be useful). Use the same procedure for installing these memory chips as for the keyboard unit, facing them in the direction of the notch on the sockets. Once again, check carefully for bent pins or pins

out of the sockets, reinstall the cover, and power up the interface and keyboard.

Press ENTER in response to MEMORY SIZE?, and your 32K machine should read 31956, and the 48K machine will read 48340 (two bytes less each in later models). Early expansion boxes, because of design flaws in memory timing, are significantly more sensitive to memory speed. If expansion memory is occasionally balky or shows frequent glitches when peripheral devices are attached, make the hardware changes to Z69 recommended in the 200% speed modification (later in this Chapter). Seeming memory failures can most often be attributed to these timing problems, although earlier interfaces (particularly those with the bulbous buffered cable) had hardware difficulties which made them extremely sensitive to noise and vibration.

Most of these earlier units had their circuit board layout and plating done in such a way that a sharp tap on the box, board, or cable would cause a 'microphonic' reaction. That is, the vibration would be transmitted along power supply and signal lines, interfering with the actual data. The result would be frequent memory crashes. Likewise, a noisy environment (nearby washers, mixers, fluorescent lights, transformers, and even printers) can cause electronic interference which would disrupt memory.

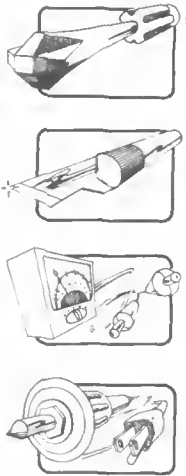
## Rescuing the RESET

Among the conveniences of the TRS-80 keyboard computer is the Reset button. A program, especially one with machine language components, may cause the computer to 'hang'. The Reset button conveniently recovers control of the machine and returns it to you.

Once the Expansion Interface is connected, though, things begin to change. The Reset button becomes a Reboot button, causing any operating programs in memory to be lost and the complete system to restart from initialization of the disk operating system (see Supplement to Chapter 1, on the power-up sequence).

As an aside, let me note that the Z-80 HALT instruction does not have the effect of a true HALT. Instead, the CPU's Halt Acknowledge output line is tied in with the Reset button. The result is a READY in Level II and a disastrous reboot with an expansion box connected. This is another good reason for reasserting the Level II reset function with this modification.

## Up-Front RESET



The solution to this problem is to disable the disk controller whenever disk access is not expected. Open the expansion box, and locate Z32, near the power switch. If you have a newer expansion interface, this circuit will be marked Z39. This is a 16-pin circuit, type 74LS155. Identify the circuit trace that runs from pin 4 underneath the IC and out the opposite side. Use an ohmmeter if necessary to make sure you have the right trace. This signal activates the disk controller chip's output to the CPU; when it is cut, the keyboard unit cannot 'see' the disk controller.

Take a sharp blade and cut this trace. Solder a 10K ohm resistor from the *far side* of this trace to pin 16 of Z32 (or Z39). This is the +5 volt lead, and will hold the pin high.

Next run a pair of fine wires from either side of the cut trace to each connection of a small toggle switch. When the switch is on, the cut trace is bridged, and the disk controller buffer can be activated normally; when the switch is off, the Reset button sends the software to a routine that checks for a disk controller. Since it does not 'see' the controller, it acts as if it were simply in Level II BASIC and returns to READY.

Be sure to mount the switch as close as possible to the trace cut, preferably right on the front of the expansion box, as shown in Photo 4-1. This will prevent noise from creeping in to an already somewhat noisy box.

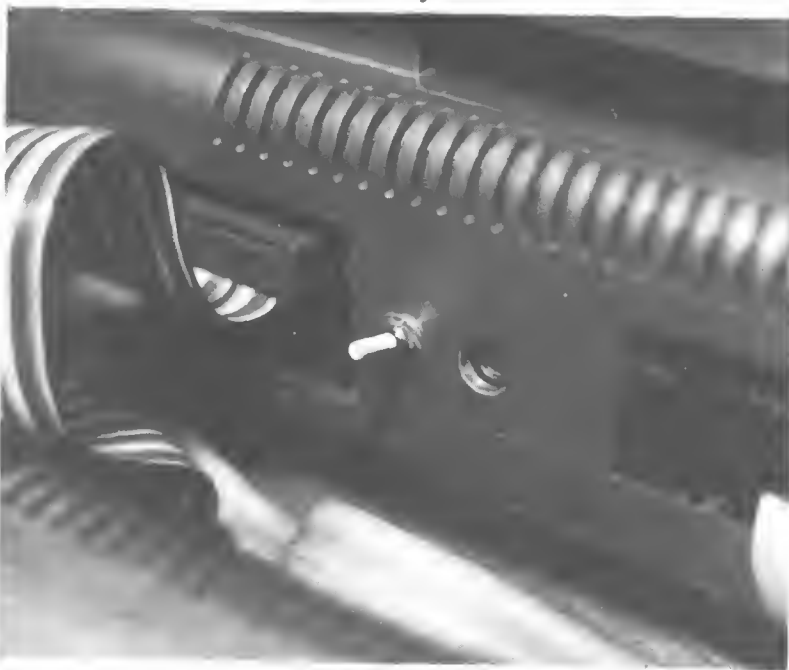


Photo 4-3. Expansion box reset modification.

Visible in this photo are a 12-inch replacement expansion cable, and the disk-defeat switch to recover the reset function.

The LNW expansion interface change would normally be identical to that for the Radio Shack box, except for the integrated circuit numbers. One additional change is necessary.

The trace from U19 pin 4 leading to U8 and U15 is cut, and a 10K resistor wired from the trace end closest to U8 and U15 to +5 volts (found at U19) pin 16).

However, the pullup/pulldown resistors in the LNW expansion box can still give an 'on' reading to the CPU. To avoid this, change the pulldown resistors from 220 ohms to 470 ohms (or, if they already are 470 ohms, from 470 ohms to 680 ohms). This will result in the 'high' reading needed to avoid picking up the disk controller signal.

## Up-Front RESET

If you are a frequent user of the Reset button in Level II — and once again a user of it with your expansion box — then you will want to get the button out of the area of the sensitive interconnect cable, and well within reach. It will be a welcome relief from clawing at the silver port cover, or totally wiping out your program by jostling the cable.

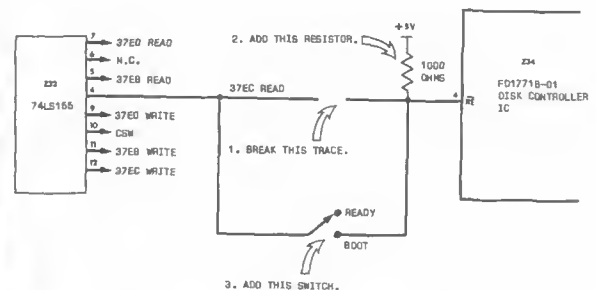


Figure 4-3. Expansion box reset modification.

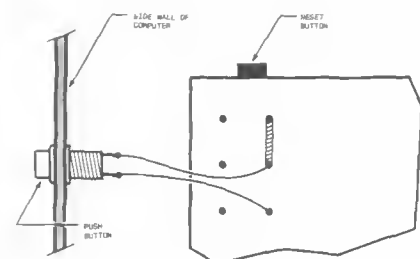


Figure 4-4. Up-front reset addition.

A momentary-on pushbutton (such as Radio Shack part number 275-1547) can be added to the cover of the keyboard unit. Photo 4-2 shows the position of the Reset button on the left side of the computer. Run two wires from the pushbutton to a small cable connector (a submini plug), and run two wires from the Reset

button (see Figure 4-2) to the other end of the cable connector (a submini jack). The project can be completed in ten minutes, carpentry and all, and reset will be less frustrating.

**Note :** Don't risk being a victim of the Apple syndrome! Apple's Reset button is placed much too close for comfort to the user's work area, and many a program has disappeared into the electronic stratosphere when inadvertently pressed while typing. So keep that Reset button just out of reach!

### Working by the Woodstove

There comes a time when sitting up straight by the computer is no longer fun. Or when the neighborhood kids howl because they can't all reach the keyboard at once during an action game. Or when those kiddie hands are just too sticky for your sacred micro. Or, in my case, when the computer's room is just too cold to share with my typing fingers. That's when you need a keyboard and monitor in the room by the woodstove. Or an extra keyboard for the young'ns. Or a keyboard for the lap in an easy chair and a monitor on the mantle.

The additional keyboard is mostly a matter of carpentry, because there's nothing special about the TRS-80 keyboard. It's merely a matrix of switches, eight by eight. Each position in the matrix is identified by the computer's software and turned into a character.

Start by obtaining two high-quality wire-wrap integrated circuit sockets, and one good solder-tail type. These are 16-pin sockets. You will also need fine wire, a 16-wire jumper cable with plug attached, and a keyboard.

The keyboard can be any style you like, from a complete alphanumeric keyboard (\$40 to \$120), to a \$10 numeric keypad if you work mostly with numbers. Whichever you choose, it must consist of individual keys, each with a single-pole, single-throw (SPST) contact pair. Many small calculators have a prearranged matrix which is incompatible with the TRS-80. If you choose a matrix keyboard, check that it will work with the TRS keyboard pattern shown in Table 4-3.

e	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	...	Unassigned...			
D	1	2	3	4	5	6	7
B	9	:	;	,	-	.	/
ENT	CLR	BRK	UPR	DNR	LFR	RTR	SPC
SHIFT	.....	Unassigned.....					

Table 4-3. Keyboard matrix.

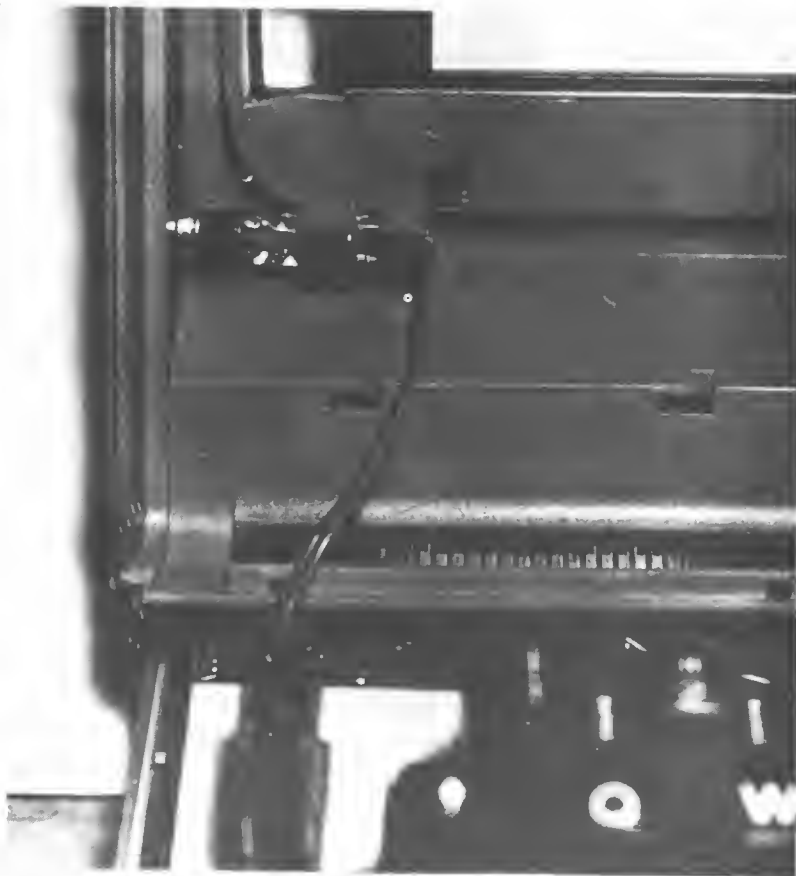
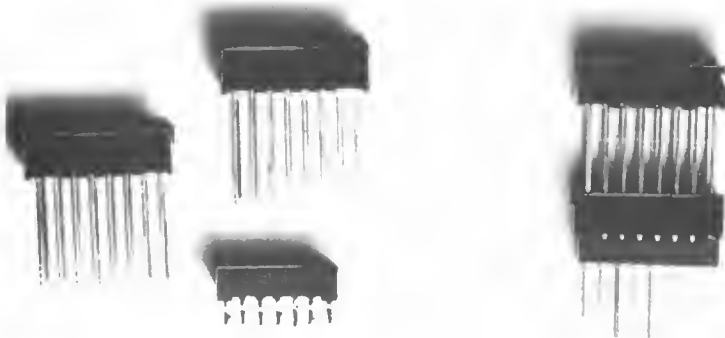


Photo 4-4. Up-front reset addition.

Reset switch added to the top cover. A connector is added so the cover can be removed easily.



*Piggybacking sequence:* pins are removed from short solder-tail socket, and it is used as a grommet for a wire-wrap socket. After this is soldered to the baseboard, a third socket is plugged in place.



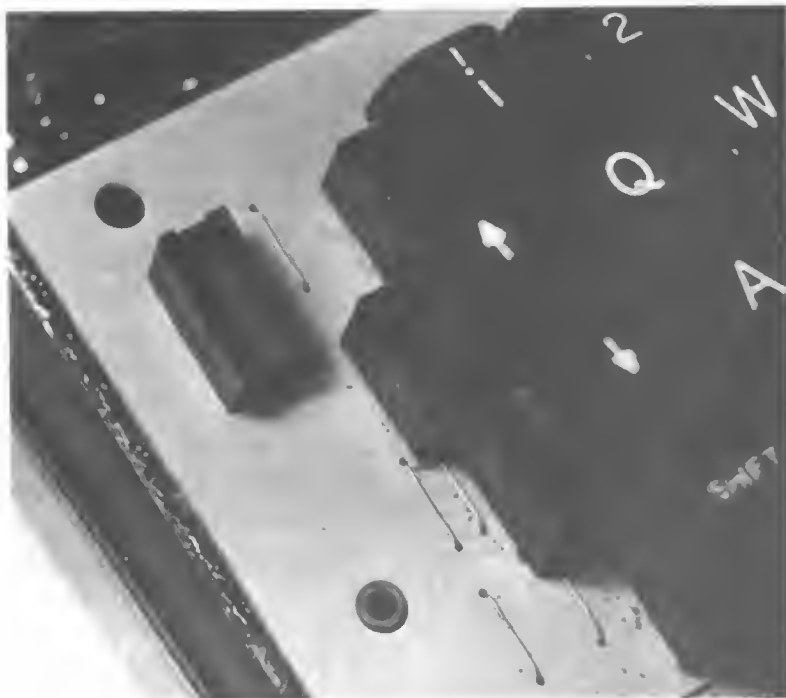
Since depressed keys are identified in software by the row and column, you only need to know which column-row combinations produce the letter you want. That way, you can reassign your keyboard's characters for any purpose that suits you - including the Dvorak keyboard. Thus, you need make no software modifications to your favorite machine language programs to use them with different keyboard combinations.

Furthermore, the attachment of a 64-key musical keyboard can open the door to direct compiling of music as you play it.

The physical layout of the TRS-80 keyboard unit is fairly compact, leaving only a space on the far left or far right for the added keyboard connection. I have chosen the left side for that addition. Inside the computer, this location is directly above a blank part of the keyboard's circuit card.

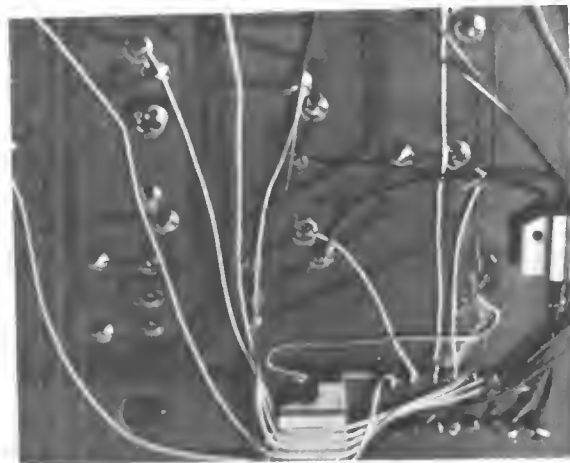
Use a strong flat screwdriver to snap out the black portion of the keyboard cover. Six tabs hold it in place at the top and bottom. Mark precisely where the free area can be found on the baseboard.

You will be using the three IC sockets to make a standoff-style keyboard connector. Pull all the pins from the solder-tail socket, and use this socket as a guide to drill 16 holes in the baseboard. Use a very fine hobby drill - #68 is good. When you have the holes completed, slide



The extra keyboard socket is soldered in place, with the gutted solder-tail socket used as a grommet.

one of the wire-wrap sockets into the disemboweled solder-tail socket, and feed the wire-wrap pins through the circuit board. Fasten with fast-drying epoxy; do not use white glue, as this will react badly with metal.



Plastic carriers from flat-pack integrated circuits make excellent 'bridges' to hold wires in place. A drop of glue holds them there.

When the glue is set, remove the entire keyboard cover, turn the baseboard over, and identify pin 1 of the newly installed socket. This pin will attach to column one of the keyboard matrix (see Table 4-4). On most versions of the TRS-80, you can use the keyboard's resistors to identify the columns; I recommend this, because there were at least three separate runs of keyboards, each with a different layout.

Column 1	R8	● H P X 0 8 ENTER SHIFT
Column 2	R5	A I Q Y 1 9 CLEAR
Column 3	R3	8 J R Z 2 : BREAK
Column 4	R2	C K S 3 ; UPARROW
Column 5	R7	O L T 4 , DOWNARROW
Column 6	R1	E M U 5 - LEFTARROW
Column 7	R4	F N V 6 . RIGHTARROW
Column 8	R6	G O W 7 / SPACE

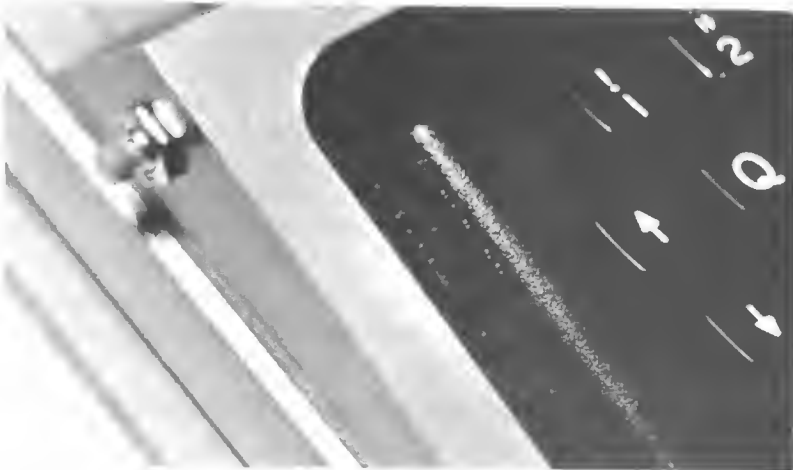
Table 4-4. Keyboard column assignments.

Match column one, then, with socket pin 1; column two with socket pin 2; column three with pin 3; etc. Solder a separate wire to each of the resistors, and wire-wrap or solder the other end to their respective socket pins. Make sure you solder to the end of the resistors which are connected to the keyswitches, not the other ends, which are all connected together.

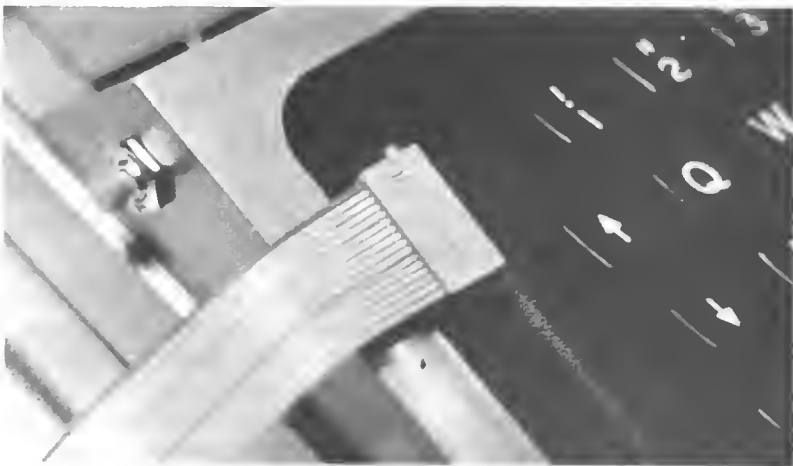
The keyboard matrix rows are found at the input pins of the on-board ICs, but because of the many versions of the TRS-80 keyboard which have been issued, this sequence is inconsistent. The technical manual identifies the rows as shown in Table 4-5, but it's better to check for yourself. Look for the traces that connect



The extra keyboard socket is soldered in place, with the gutted solder-tail socket used as a grommet.



Socket rises perfectly to the height of the outer shell. Note reset button extension at left.



Completed extension cable plugs discreetly into the socket. Any type of keyboard, from a small numeric pad to a full 64-key musical keyboard, can be used.

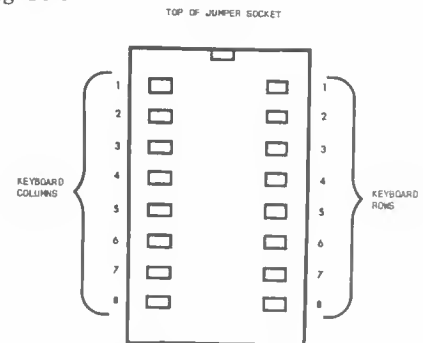
together @, A, B, C, D, E, F and G. These are all in row one. Solder a wire to some point in this row, and run it to pin 16 of the new keyboard socket.

Row 1	Z1 Pin 8	@ A B C D E F G
Row 2	Z1 Pin 2	H I J K L M N D
Row 3	Z1 Pin 10	P Q R S T U V W
Row 4	Z2 Pin 2	X Y Z
Row 5	Z1 Pin 6	0 1 2 3 4 5 6 7
Row 6	Z1 Pin 4	8 9 ; , - . /
Row 7	Z1 Pin 12	ENTER CLEAR BREAK UPARROW DOWNARROW LEFTARROW RIGHTARROW SPACE
Row 8	Z2 Pin 4	SHIFT

Table 4-5. Keyboard row assignments.

Locate row two by using Table 4-5, and solder a wire from somewhere in this row to pin 15. Likewise, identify rows three through eight, and solder them to pins 14, 13, 12, 11, 10 and 9, in order. When viewed from the top, the pin arrangement is as shown in Figure 4-4.

Figure 4-4.



Once the wiring is complete, clip the pins on the added socket very short, turn the board over, and put everything back in the case. Power up and check the operation of the computer.

Now clip a small length of bare wire about an inch long, and bend it in the shape of a 'U'. At the newly installed socket, jumper each row across to each column, one at a time. You should produce all the non-shifted keyboard characters on the screen, including the previously inaccessible four arrows and the cursor character.

Clip an additional jumper, and cross row eight with column one. This simulates the pressing of the SHIFT key. Repeat the column-row jumpering, and note that all the shifted characters now appear. Any unusual behavior, such as repeated letters or groups of unrelated letters produced from a single jumpering, indicates a wire may be shorted, attached to the wrong column or row, or left out completely.

Finally, as with all modifications, make the cosmetics pretty. Snap the black plastic cover off again, and in it cut a rectangular hole the size

of the 16-pin socket, using a hot, sharp X-acto knife or razor blade. Work slowly, filing or smoothing, and rub the finished hole with a marble. This will result in a professional-looking addition.

The second wire-wrap socket now piggybacks into the first one, and the black cover snaps back on. The socket should fit perfectly, rising about 1/16 inch above the surface of the cover. The 16-wire cable plugs into it a comfortable distance from the typing area, well above and to the left of the up-arrow key.

For each keyboard you wish to add, work out the row-column matrix using the table. A jumper cable may be an integral part of each keyboard, or an IC socket/lug arrangement similar to the main unit can be included with each added keyboard. You can even chain keyboard to keyboard by including two sockets on each one — just be sure all the sockets and plugs are identically wired!

## Working by the Woodstove — II

Once you've got a new keyboard in your lap, you'll probably want a nearby screen to glance at. There are two ways to do this: by using a video monitor or by using an ordinary television.

There are advantages and disadvantages to both methods.

A video monitor is the ideal tool because the image is crisp and clear, and your TRS-80 provides a 'composite video' (video with both image and synchronization signals) output. But a video monitor also costs somewhat more than a new black and white television, and means an added expense in any case.

A television on the other hand has limited 'bandwidth'; that is, it was made for fluctuating images, and not for the precise on-off quality of white letters on a dark background. If you've ever noticed that it's sometimes hard to read telephone numbers, addresses, or credits for television programs, you've got an idea of how hard it can be for some sets to reproduce crisp computer lettering.

Furthermore, most televisions accept only radio frequency (RF) input, meaning your TRS-80 output has to be converted to RF before your television can make sense of it. The last complication is that such a close and strong RF signal can overload your television's automatic gain control (AGC) resulting in an unstable, twisting, rolling, or badly contrasted picture.

But chances are you already have a television, and chances are even better that the television is

### Making it Look Manufactured

One of the worst curses of a customized anything is how it tends to look — homemade. Now I have absolutely nothing against something looking homemade, but whenever I do that, somehow it also *acts* homemade — that is, just a bit too eccentric to be reliable as a computer!

Instead, attend to the cosmetic aspects of the TRS-80. Since the plastic case is very pliable and 'works' easily, these touches are easy. The silver coloring is a flake paint, and tends to wear off, particularly below the shift keys where the typing hands rest. The black plastic cover is very soft and can be scratched; its pebbled surface makes such scratches stand out.

If the silver flake paint wears off, it can be resprayed with the kind of paint used on model cars. Work the spray can valve for a while until it is spraying evenly, and then spray the silver cover from a slight distance. If the color match is not perfect, the bottom can be sprayed as

well. Use two or three extremely light coats for a good effect.

The black plastic cover can even have deep, obvious scratches repaired by rubbing it with a glass marble. The scratch will smooth over, making it different from the area around it. Next, pick at the smooth area very lightly with a needle, making pebble-size marks similar to the rest of the case. Rub with the marble again until the scratched area looks exactly like the surrounding area. It really works.

When cutting holes for switches, buttons, sockets, jacks, keypads, etc., always cut the holes slightly smaller and use rattail (for round holes) or triangular (for rectangular holes) file to expand it to the correct size. This way, no unsightly cut marks will extend away from the area of the modification. Bevel cut rectangular surfaces with a flat file, smooth them with a letter opener or librarian's 'bone', and touch up the corners. The result will be almost precisely like the manufacturer's molded cutouts.

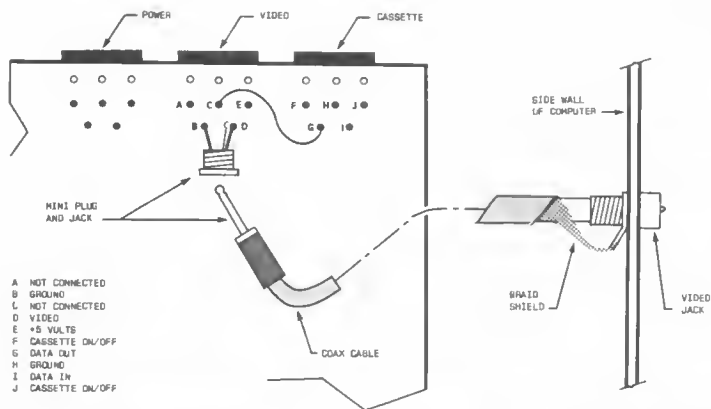


Figure 4-5. Extra video jack.



Photo 4-7. RF modulator hookup.

Additional jacks added in order to feed both the video monitor and an RF modulator installed in the expansion box.

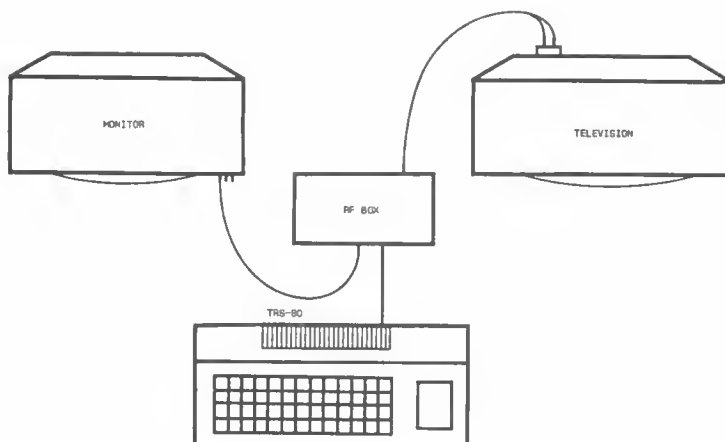


Figure 4-6. RF modulator hookup.

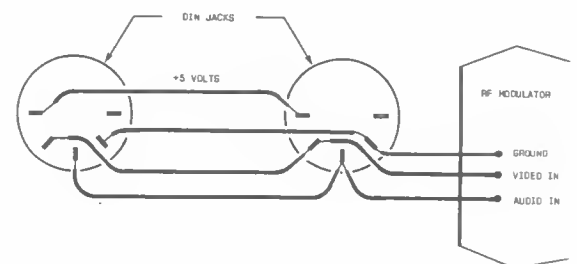


Photo 4-6. Extra video jack.

just where you were thinking of using your computer.

Before going on with these modifications, there is one important note. Working with any unknown television or monitor can crash your system if it is electrically noisy. If you plan to send a video signal directly to a monitor, or especially to a television modified for video input, make sure the set isn't 'hot' – no AC line voltage should be floating on the case. This can damage your computer . . . or you! If you are considering a monitor or direct video input, and you are not familiar with your video sets, then take the television or monitor to a service person who can check them out. With appliance devices such as ordinary televisions, this is doubly important. This note does *not* apply if you will use RF input to a television.

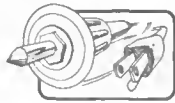
Whichever method of added video you decide upon, though, there is a solution. The first and easiest is to add an extra video output jack to your computer. There's plenty of video signal to be had, and it can be shared among several sets. Figure 4-5 shows how to wire that extra jack, and Photo 4-6 shows how mine is installed. The



## Hexadecimal Keypad



connector shown in the photo is a miniature Amphenol connector used for microphone cable, although any kind of microphone, video, CB, phono, or other shielded coaxial cable and connector can be used.



A second approach is to send the computer's video signal to an RF generator, and feed that to your television. An RF modulator is available in kit form from Radio Shack (part number 277-122, with TRS-80 installation instructions) and also from other suppliers. A few plugs and jacks are needed to complete the job.



Assemble the kit or purchase a surplus modulator, and hook it to an ordinary television. Either run the new video output to the modulator's input (use the directions with your modulator), or install a separate DIN plug-and-jack pair as in Photo 4-7.).



Actually modifying an ordinary television for video input is tricky, and I won't cover the topic here. If you are interested in doing this — and picture quality will be very much improved — refer to Don Lancaster's *TV Typewriter Cookbook*.

The results of your video modifications will be dependent on how clean your soldering is, the layout of your wiring, etc. If you add the RF modulator, you may notice something similar to herringbone on your TRS-80 monitor if the

contrast is turned all the way up. Move the modulator away, or put it in a shielded (all metal) box, and the herringbone will disappear.

If you use a long cable directly from the video output, there will be a huskiness to the characters on your TRS-80 monitor. This is actually a kind of 'smearing' introduced by the capacitance of a long cable. This is not bothersome to me; in fact, it actually seems to improve the clarity and boldness of the screen characters.

Finally, if your RF-input television addition results in an unpleasant display on the TV, try to adjust the automatic gain control (AGC) on the back of the set. Tune it in carefully as well. There will probably be less clarity in the 64-character mode than you are used to with your monitor, unless you have a very good set.

## Hexadecimal Keypad

Entering machine language programs using T-Bug or another monitor is tedious enough without having to search all over the QWERTY keyboard for hexadecimal numbers. Instead, a keyboard can be added right onto the TRS-80. If you have a numeric keypad included with your Level II unit, you might want to remove it to add this one.

The addition of a hexadecimal keypad is mostly carpentry, since the connections are made in parallel to the main board, exactly like those connected to the socket addition described earlier. An unencoded hexadecimal keypad to do the job is available from Jameco Electronics (see Appendix 1. ), and one of its keys can be set aside for an *Electric Pencil* or other control key, adding significant programming power to your custom TRS-80.

For this modification, you will need two 10-inch strips of 1/2-inch by 1/2-inch plastic rod (plexiglas or lucite are best, but wood strips will work as well), five-minute epoxy, wire, and the notorious hot razor blade and marble for the cosmetics.

Undo the cabinet as usual, and take the entire electronics out of the case. Later TRS-80's have an on-board, two-chip Level II ROM set, but if your Level II ROMs are the type on a separate board fastened to the end of an interconnect cable, then they will have to be moved. They are fastened to the bottom right of the circuit card with double-face tape; slit the center of the tape with a razor blade. Do not pull the ROM board off by force, as the pressure might crack either circuit board.



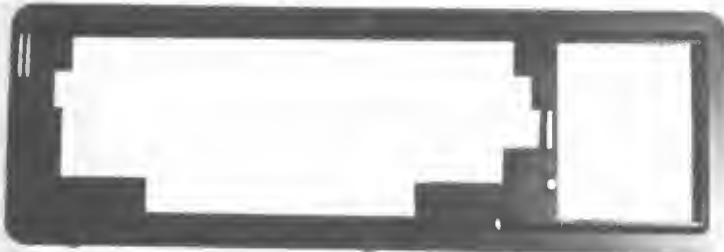
Photo 4-8. An Extra system near the woodstove.

The kitchen installation. An old RCA television and a surplus keyboard make working with the TRS-80 more convenient. Computer remains in one place, but extra monitor and keyboard can be moved closed to the woodstove on cold days. The woodstove offers no electronic interference.

The interconnect cable to the ROMs is long enough so they can be remounted inside one of the case 'feet', or above the hexadecimal keypad. Pick up a small piece of double-face tape to refasten them, or roll masking tape into cylinders (remember hanging pictures in fourth grade?).

The Jameco Electronics keypad base is identical to the TRS-80's in height and depth, so

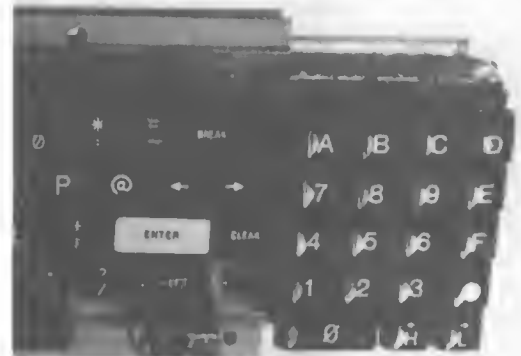
**Photo 4-10.** *Black plastic cover used as template.*



**Photo 4-10.** *Black plastic cover used as template.*

*The black plastic keyboard cover can be snapped out and cut to fit. A paper template is used first to verify the position of the hex keypad.*

the two plastic strips can be used to create a 'trailer hitch' arrangement with the smaller keyboard. Support both boards firmly so that they are parallel and the hex pad meets the TRS-80 printed circuit base. Cement the plastic strips in place with the quick-setting epoxy, and make sure the vertical alignment of both keyboards is identical. If you have a later style



*Keyboard is attached with runners to the main keyboard, and glued in place with epoxy.*

## Cleaning the Keyboard

If your TRS has the old-style keyboard that was badly afflicted with keybounce, there are many ways to take care of it other than perennially loading a KBFIX routine.

The first rule is to *make sure you have the old style keyboard!* The newer keyboards have a sculptured curve to their arrangement when viewed from the side. These new boards have a contact arrangement which can be destroyed by trying to remove the keycaps. But these keyboards don't have a keybounce problem anyway.

Bend a paper clip into an ingenious keycap lifting tool, like this:



**Figure 4-7.** *Keycap lifting tool.*

Slip this clip underneath a keycap, and lift up. The plastic cap will come off, revealing a hole in which two metal plates are protruding upward.

The best way is to test the keys. Press each key quickly, gently, slowly, or sharply, until you decide whether it is a 'bouncy' key. If it still bounces after the cleaning, then take a hatpin (are there still hatpins?) or heavy sewing machine needle, and push the tines in line. This is a very delicate job; use caution and a magnifying lens. If the plates are not parallel to each other, or if they are vertically misaligned, use the pin to shift their positions.

The villains are dirt and bent tines on the plates. Brush out the dust, dirt, or hair (or much better, blow it out, using photographers' compressed air, such as 'Dust-Off'). You will be amazed at the cloud of grit that rises from the keyboard. Next, examine the tines of each key very carefully. They should be perfectly in line, so that when a key is pressed, all come into contact with their opposite (un-tined) plate.

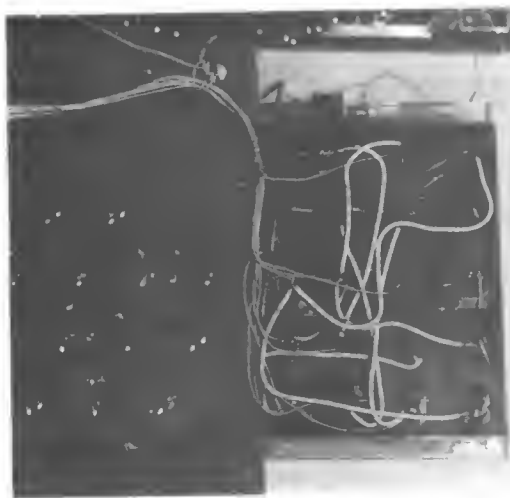
Check all the keys for bounce again, and work until it is completely cured. Some folks recommend a spray of contact cleaner at this point; I recommend against it. The cleaner tends to stay wet for a while, and dust and grit can get back into it very quickly, collecting into a dusty mudpile. Instead, give all the keys a last brushing or spraying with compressed air, and fit the keycaps back on.

Keybounce should be gone for quite a while. Monthly cleaning will keep the keyboard in shape.

## Hexadecimal Keypad

keyboard (with the curved keyboard array), you will have to adjust the carpentry slightly. You may also experience a bit of keybounce on the new keypad unless you keep the keys clean.

When the glue has set, use the black plastic cover as a template for drawing your current key positions and, with the aid of a straightedge, draw extension lines horizontally across that drawing. These are the upper and lower limits of the new keypad opening. Align the template with the complete alpha/hex keyboard assembly, and mark the vertical positioning of the hex keys, allowing about 1/32 inch additional on both sides for key-travel room. This will bring you within



*Back of keyboard is wired by soldering directly to the key pins. Wires are then run to resistors and integrated circuits found on the main keyboard.*

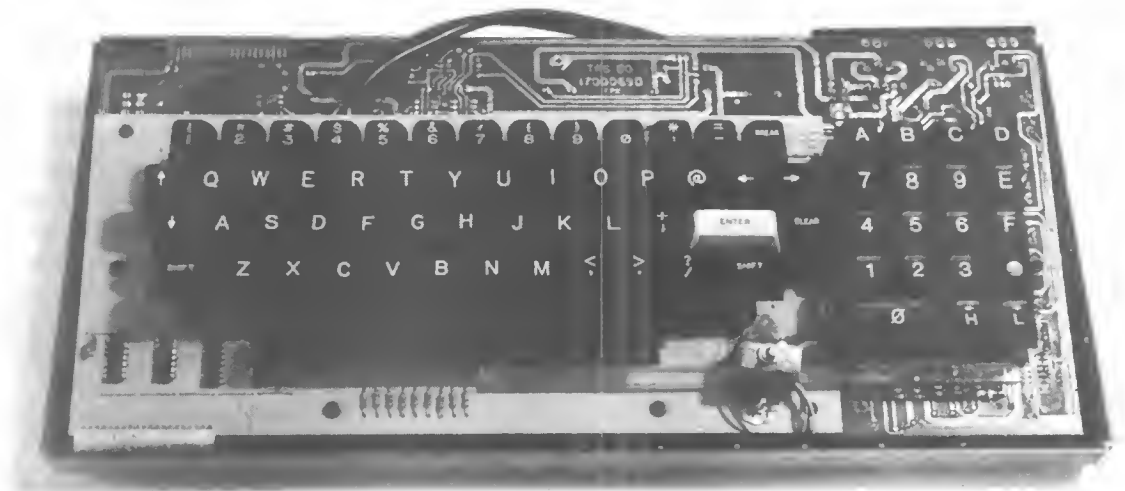
1/4 inch of the power LED.

Using the template with the black plastic cover, carefully cut an opening in the cover with the hot razor blade. This is the most time-consuming task, and should be accurate enough for the keys to travel easily (fit it atop the keyboard before re-installing the modified computer), and should look factory-finished.

It's about time to interconnect the wires from the hex pad to the main keyboard, but before that, you'll probably want to rearrange the keycaps on the hex pad. Using the lifting tool, pull off the keycaps and put them in a convenient order for hex entry; I used the accounting arrangement, bottom to top. This is the pattern used for the wiring arrangement shown in Diagrams 4-1 and 4-2.

Rest the keyboard on its face, and separate it gently from the main circuit card. Set the keyboards in an accessible position, and solder fine wires (wire-wrap type is easiest to use) to the hex pad connections shown in Diagram 4-1. Route the individual wires from the hex keys to the points on the circuit card shown in Table 4-5. As before, check to make sure this IC arrangement matches your board.

Next, solder wires to the hex pad contacts as shown in Figure 4-1, and route these wires from the hex keys to the circuit card's resistors noted in Table 4-4. Double check these too against your version of the keyboard. Once both sets of wires have been run, gather them in neat hanks and fasten them along their routes with wire ties (plastic bag ties will also work well).



*Completed keyboard seats easily in the case bottom. A piece of insulating plastic should be inserted under the added keyboard to prevent pushing the keyboard pins into the main circuit board and causing a short.*

## Hexadecimal Keypad

Table 4-3

⓪	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	....Unassigned....				
0	1	2	3	4	5	6	7
8	9	:	:	,	-	.	/
ENT	CLR	BRK	UPR	ONR	LFR	RTR	SPC
SHIFT	.....Unassigned.....						

Table 4-4

Column 1	R8	⓪	H	P	X	0	9	ENTER	SHIFT
Column 2	R5	A	I	Q	Y	1	9	CLEAR	
Column 3	R3	B	J	R	Z	2	:	BREAK	
Column 4	R2	C	K	S	3	;	UPARROW		
Column 5	R7	O	L	T	4	,	DOWNARROW		
Column 6	R1	E	M	U	5	-	LEFTARROW		
Column 7	R4	F	N	V	6	.	RIGHTARROW		
Column 8	R6	G	O	W	7	/	SPACE		

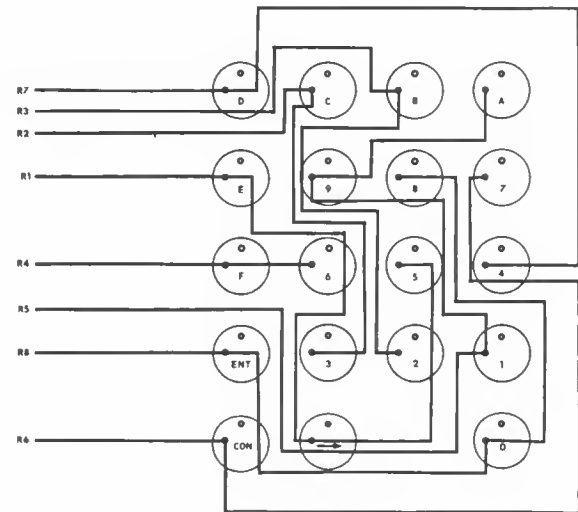


DIAGRAM 4-1

Table 4-5

Row 1	Z1 Pin 8	⓪	A	B	C	D	E	F	G
Row 2	Z1 Pin 2	H	I	J	K	L	M	N	O
Row 3	Z1 Pin 10	P	Q	R	S	T	U	V	W
Row 4	Z2 Pin 2	X	Y	Z					
Row 5	Z1 Pin 5	0	1	2	3	4	5	6	7
Row 6	Z1 Pin 4	8	9	:	:	,	-	.	/
Row 7	Z1 Pin 12	ENTER	CLEAR	BREAK	UPARROW	DOWNARROW	LEFTARROW	RIGHTARROW	SPACE
Row 8	Z2 Pin 4	SHIFT							

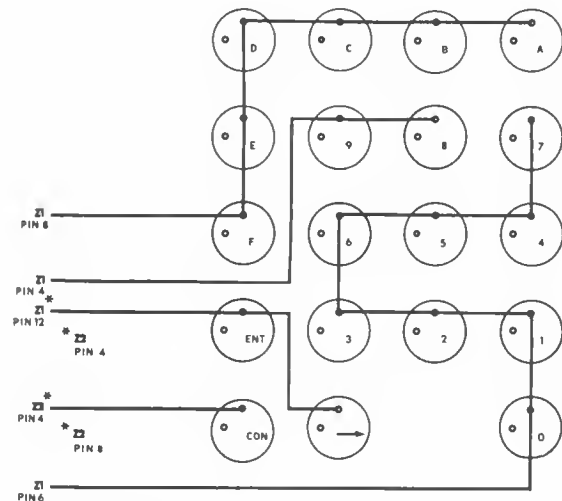


DIAGRAM 4-2

## Hexadecimal Keypad



*Level II ROM board can be fit immediately above the hex pad. Insulating double-face tape can be used to hold them in place.*

Reassemble the keyboard in its case, remembering to orient the Level II ROM board safely in its new position. Reinsert all cables, and restore power. The keys on the main keyboard should respond normally; check them all. Now check the keys on the hex pad. All but the bottom right one should have an effect. To test its operation, enter this program:

```
10 CLS
20 PRINT PEEK (14464);
30 GOTO 20
```

The value you read should be zero unless either the shift key on the main board or the bottom right hand key on the hex pad is depressed. The shift key will return a value of 1; the new key will display a value of 16. Pressed together, they will read 17.

All keys should now be working properly. As with the socket addition described above, problems will occur in the form of incorrect letters, groups of letters on a single keypress, or dead keys. If any of these symptoms appear, recheck for shorted or unconnected wiring, or a difference in your model TRS-80 keyboard. Refer to Tables 4-3 through 4-5 if you suspect the latter.



*The final modification with black plastic cover fit back in place looks manufactured (almost).*

## Reversing the Video

The video display of most computers including the TRS-80 suffers from a tremendous flaw – contrary to what we have known since we first learned to read, the letters are presented to us in glowing blue-white on a black background. Serious use of a computer as a day-to-day appliance, as a true adjunct to our daily lives, is limited by its formidably unappealing, tiresome, and illegible display.

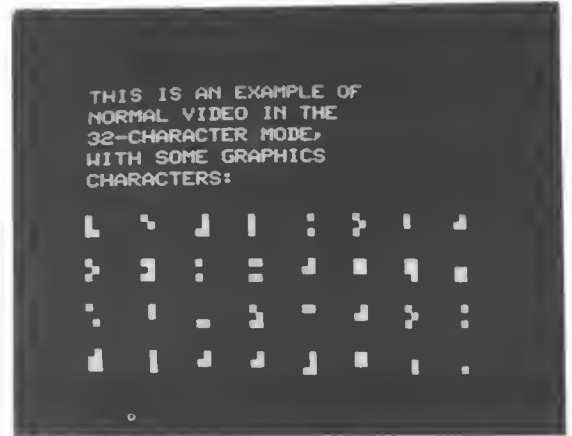
The reverse video modification is surprisingly easy, so the question arises: why is it not a standard feature of small computers? The answer is found partly in a tradition of computer monitors which have always been lighted characters on a dark background. The remainder of the answer is found in the video display itself, which is more often than not incapable of presenting a legible character in the black-on-white mode.

This problem arises with the TRS-80 as well; the video monitor has weaknesses which are emphasized by reversing the characters. But overall, and with a small change to the monitor itself, the display can be made quite legible and easy on the eyes.

For this modification, you will need three integrated circuits: 74LS02, 74LS74, 74LS368.

A 1.5K-ohm resistor will also be used, and wire-wrap wire for the interconnections. There are two ways of making this modification: on a separate board, or piggybacked atop chips already present inside the TRS-80. Since the latter approach would involve at least 15 separate wires, both this change and the high-speed modification presented below will use the piggyback system.

Open the computer's case, and remove the



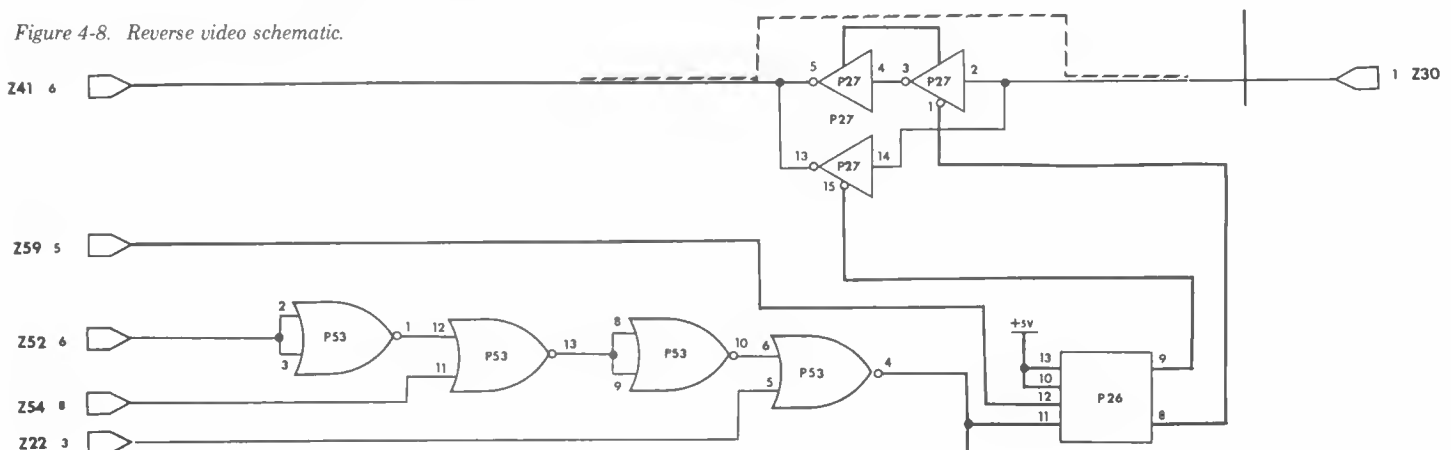
*Normal screen has unnatural bright characters against a dark background, which is wearying on the eyes.*

electronics so the integrated circuit side of the main card is up. Locate the following circuits (the numbers are silkscreened on the board): Z22, Z26, Z27, Z30, Z52, Z53, Z54, and Z59.

The output of Z54 (pin 8) is a decoded signal representing ports 254 and 255. Combined directly with the output of Z52 (pin 6), port 255 is selected for use by the cassette and video circuits. By inverting the output of Z52 and combining it with the output of Z54 and the computer's OUT signal (Z22, pin 3), port 254 can be selected. Figure 4-8 is the schematic for this complete decoding process.

The decoder uses the 74LS02. Prepare the integrated circuit by bending all the leads except pins 7 and 14 so that they are parallel with the IC's body. Locate Z53 on the TRS-80. Seat the 74LS02 directly atop Z53, with both notches or dots facing in the same direction. Solder power pins 7 and 14 of the piggybacked IC to corresponding pins 7 and 14 of the one below. I will refer to this piggybacked IC as ZPORT.

Figure 4-8. Reverse video schematic.



Find Z52. Run a wire from pin 6 of Z52 to both pins 2 and 3 of ZPORT, and solder it. Next locate Z54. Solder a wire from its pin 8 to pin 11 of ZPORT. Finally run a wire between pins 1 and 12 of ZPORT.

As mentioned, Z22 contains the needed OUT signal. Run a wire from pin 3 of this circuit to pin 5 of ZPORT. Solder together pins 8, 9 and 13 of ZPORT, and run a wire between pins 6 and 10 of ZPORT. Pin 4 remains unused, and it contains the complete decoded signal of port 254 as shown in Figure (?). The BASIC command OUT 254,N will activate this signal.

The next IC to be prepared is the 74LS74. Again, bend all leads parallel to the body except 7 and 14, seat this upon Z26, and solder the power pins (7 and 14) in place. This piggybacked circuit I will call ZFLOP, as it will determine which state (normal or reverse video) is flip-flopped into place when OUT 254,N is commanded.

Run a wire from the decoded signal at pin 4 of ZPORT to pin 11 of ZFLOP. Z59 has a convenient data line (bit 1) at its pin 5; run a wire from there to pin 12 of ZFLOP. Now run short wires connecting together pins 10, 13, and 14 of ZFLOP. With these connections made, OUT 254,0 will flip the circuit, and OUT 254,2 will flip it. (Is the suspense building?)

The final IC is now prepared. Bend the leads of the 74LS368 parallel to its body, except for pins 8 and 16. Seat this on Z27 and solder power pins 8 and 16 to it. For convenience, this piggybacked circuit will be called ZMODE.

Find Z30 and Z41. Pin 1 of Z30 is connected via a circuit board trace on the *underside* of the board to pins 6 and 7 of Z41. Z30 provides the characters to be output to Z41, which is part of a circuit that mixes in the synchronization information to produce 'composite video'. Cut this trace near Z30.

Why cut this trace? The TRS-80 produces characters on the screen by turning on 'dots' as the electron beam sweeps across the tube. Each dot is part of a continuous stream of pulses which might be called 'dots' and 'undots' – ones and zeros. To reverse the video, then, all you need to do is to turn the 'dots' into 'undots', and turn the 'undots' into 'dots'. We insert an electronic fork in the road at the output of Z30. When directed toward one side of the fork, the characters are made up of dots; when directed down the other fork, the characters become undots, and the background becomes dots.

Run a wire between pin 1 of Z30 and pins 2 and 14 of ZMODE. These are two inputs of an inverting buffer, the 'fork in the road'. If we invert the signal once, the video reverses . . . invert it twice, and the signal returns to normal. Connect pins 3 and 4 of ZMODE together to perform the double inversion. Pin 5 is the normal video output, and pin 13 becomes the reversed output. Connect both these outputs (pins 5 and 13) together.

Find the opposite side of the broken trace from Z30, and follow it to a hole that is plated through the board; it is at the end of a row outlined by Z29 and Z30. Be careful to select the correct hole. Run a wire from this hole to pins 5 and 13 of ZMODE. This connection feeds both normal and reverse video back into Z41 and through to the video output jack.

I have chosen ZMODE because it is a three-state circuit. That is, its outputs can be made electronically invisible. Otherwise, both normal and reverse video would be output at the same time. To choose between them, ZFLOP is used to *enable* one or the other of those outputs. Run a wire from pin 8 of ZFLOP to pin 15 of ZMODE. Run a wire from pin 9 of ZFLOP to pin 1 of ZMODE.

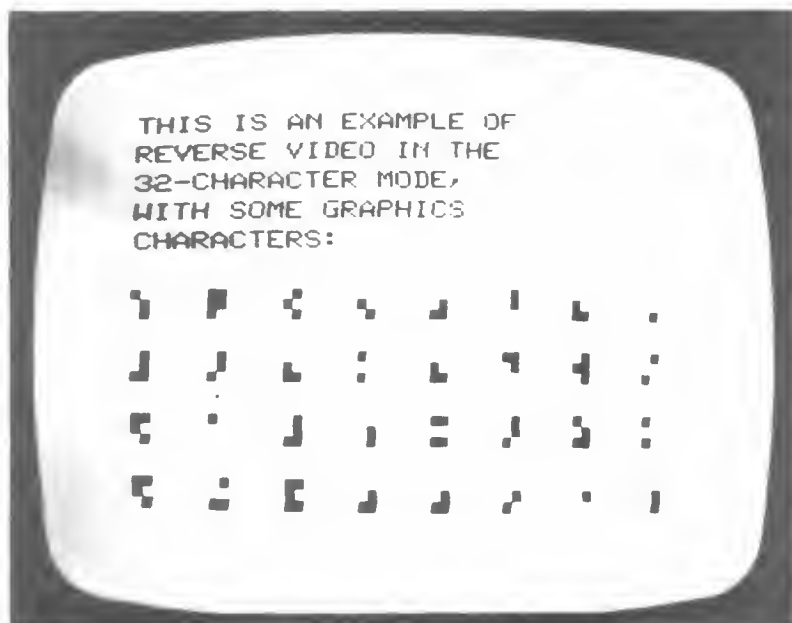


Photo 4-11. Reverse video screen example.

*Illuminated background with dark characters is clear and, together with a green screen of some type, much more gentle to look at over long periods.*

By commanding OUT 254,2 or OUT 254,0, data line 1 selects which output of ZFLOP will be enabled, and which video mode will be visible on the screen

Check your wiring and restore the computer to its case. Power up, and command OUT 254,2. The screen will reverse. The effect, alas, will not be as dramatic as you might expect because the video monitor is not a great piece of work.

Power down the system, unplug the monitor, and open it up. You might find that a hex-nut driver is necessary to open the monitor instead of a Phillips type; later monitors used 1/4-inch hex nuts.

When it is open, find the plug-in circuit card closest to the monitor wall (some have only one), and locate the resistor marked R14. The present value should be 3.3K ohms (orange, orange, red, silver or gold). We want to give the video signal a bit more 'oomph', so piggyback the 1.5K ohm resistor atop R14 and solder it in place. Restore the cabinet. The reverse video modification is complete.



### Clumsy? Me Too. Do This First.

During the course of these modifications, there will be connections made to integrated circuits which are carrying piggies atop their back. In case your soldering iron is a bit obese, you might want to solder wires to these circuits in advance.

All these pins will have wires running to them:

- Z22, pin 3
- +Z25, pins 11, 12 and 13
- +Z27, pins 4 and 5
- Z30, pins 1 and 13
- Z41, pin 6
- Z52, pin 6
- Z54, pin 8
- Z59, pin 5
- Z60, pins 4 and 5
- Z63, pin 12

Only those marked with a plus sign (+) actually carry piggies, but since everything is close together, you might want to make all the connections in advance anyway; so there they are.



Photo 4-9. Two photos (a) & (b) of two keyboards.

### Lower Case with Upper

Many modifications have been proposed to obtain lower case characters already present in the TRS-80 character generator. Some are incompatible with each other, although they do provide access to a group of special control characters also burned into the character generator.

The modification provided here is simple, compatible with both the Radio Shack and Electric Pencil modifications, and should give no grief throughout its life. For this modification you will need a single integrated circuit, and 2102 AN-4L memory chip. These chips are available from several suppliers; in a pinch, the 21L02 sold by Radio Shack will do the job.

The 2102 will be piggybacked – except for pins 11 and 12 – atop Z45, also a video memory chip (it will not necessarily be marked a 2102, since Radio Shack ordered house numbered parts for a while, but it is a 2102). Bend pins 11 and 12 of the piggyback 2102 parallel with its body, and fit the integrated circuit on Z45, making sure it is positioned in the correct direction. Solder extremely carefully, pin for pin, down one side (pins 1 to 8), and up the other (pins 9 to 10, 13 to 16).

Now locate Z25. It is an integrated circuit containing four OR-gates, one of which is not used. With solder-wick, suck up the extraneous solder that is present on pins 11, 12, and 13, and with a sharp X-acto knife or razor blade, cut pins 12 and 13 free from each other and from the ground lead going to pin 7.

Now locate the trace on the circuit board running from Z60 pin 4 to Z30 pin 13. Use an ohmmeter if necessary to make sure you have the correct trace. Cut it through. Double check that you have not cut the trace that runs from Z60 pin 4 to Z27 pin 13.

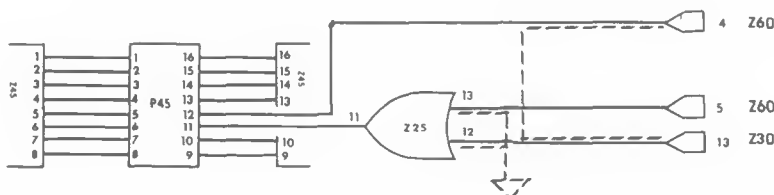


Figure 4-9. Upper/lowercase schematic.

Finally, run four wires: from Z60 pin 4 to piggy-Z45 (call it ZMEM from now on); from Z60 pin 5 to Z25 pin 13; from Z30 pin 13 to Z25 pin 12; and from Z25 pin 11 to ZMEM pin 11.

The lowercase modification is complete. A memory chip to represent the missing bit 6 has been added, and the necessary triggering information (bit 7 and bit 5) has been gated through Z25. Why is it necessary to gate bits 7 and 5 to trigger ZMEM? Because the information sent by Radio Shack's print routine does not include bit 6; in an unmodified machine the bit is generated by the presence of bit 7 and bit 5! Check the *Technical Reference Handbook* for details. In this case, the false bit 6 is generated where necessary by Z30, and ORed with a true bit 6. In other words, either case will result in bit 6 being embedded in the new bit 6 video RAM, ZMEM.

**Note:** Several TRS-80's in which I have installed this modification contained a Z25 with the spare gate dead. Perhaps it was a Radio Shack economy move. If so, piggyback a 74LS32 on Z25, soldering pins 7 and 14 to Z25, and continue with the remaining lowercase mod instructions.

The lowercase modification can be tested as follows; a complete lowercase driver is presented in Chapter 3:

```
10 CLS
20 FOR X = 15360 TO 15360+255
30 POKE X,Y
40 Y=Y+1
50 NEXT
60 GOTO 60
```

### One by One

Reversing individual characters is similar to the process of reversing the entire video screen, except that the reversal is carried out only for a short period of time. That period is determined, of course, by the letters being displayed. (At this point, I should note that all four internal hardware modifications in this chapter are partly interrelated. Both reverse video and high-speed modifications use the same output port decoding; both reverse video and reverse characters use the same flip-flop; both reverse characters and upper/lower case use the same video memory decoding hardware).

Before installing the individual-character reverse video, the upper/lower case modification must be installed. This will provide the essential bits 6 and 7 (see that section for an

explanation of the 'phantom' bit 6). For the individual character reverse, three additional integrated circuits will be needed: one 74LS86 Exclusive-OR gate, one 74LS10 triple-input NAND gate, and one 74C04 hex inverter. The last is very important, because it is used for aligning the reversal pattern with the letter to be reversed. You will also need a small variable resistor (trimmer potentiometer, or 'trimpot'), approximately 50,000 to 100,000 ohms; two capacitors, one 330 picofarads (pF) and one 0.033 microfarads (mF).

As before, bend all pins of the 74LS86 except pins 7 and 14 parallel with the body of the integrated circuit. Affix the 74LS86 atop Z24, and solder power pins 7 and 14 to it. This gate will multiplex the combined bit 6/7 signal with the output of ZFLOP (from the previous complete reverse video modification). Call this gate ZMUXX.

Remove the wires attached to the outputs (pins 8 and 9) of ZFLOP, and run them,

respectively, to pins 2 and 5 of ZMUXX. Now run a pair of wires from pins 6 and 3 of ZMUXX to pins 15 and 1 of ZMODE (which were just disconnected from ZFLOP). In other words, ZMUXX has been inserted between ZFLOP and ZMODE.

Next position the 74LS10 correctly atop Z25 and solder the power pins in place. Call this circuit ZBITS, because it will evaluate which bits of given letters should cause the reversal to take place.

The steps below summarize all the activities to complete this modification; some have been done already, (such as the lowercase modifications) but are included for clarity:

1. Break trace from Z30, pin 13 to Z60, pin 4.
2. Break trace from Z63, pin 12 to Z42, pin 13.
3. Break trace from Z42, pin 12 to Z27, pin 4.
4. Cut loose Z25 pins 11, 12, 13, from pin 7.

### Carpentry Considerations

When you first open your TRS case, you'll probably be unfamiliar with what comes out. The photo below is presented to remind you that six screws and five white spacers make up that group:

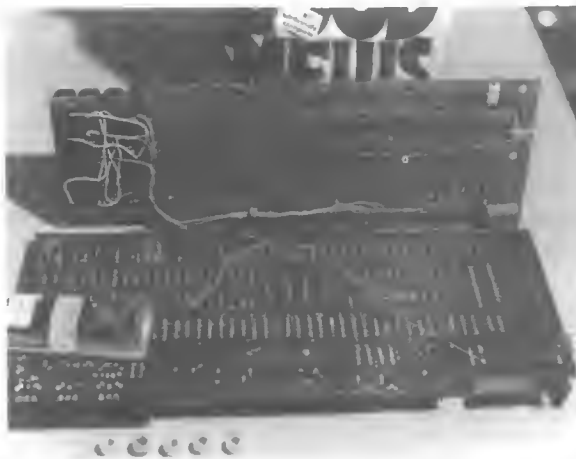


Photo 4-12. Screws and spacers in keyboard unit.

When you have been soldering in and around the circuit board, chances are that a lot of ugly, crusty brown flux residue will build up. In order to see what you are doing and make sure connections are sound, you should clean this mess. There are flux removal compounds available, and these should be applied with cotton swabs.

My own choice is a gentle but very fast acting substance which can sometimes be found in surplus – Thermo-Fax brand belt cleaner. This treats the boards and their coating without harsh chemical action, but removes the flux within seconds.

Another area of difficulty in doing these modifications is cutting traces. Place the circuit board on a very secure and stable table, cushioned just a little with a towel. Lean firmly but gently on the board, and move an X-acto knife or single-edged razor blade back and forth until the trace gives way. This may take as many as 20 or 30 scrapings.

When the trace looks cut, make sure. Cut deep into the fiberglass base so you can see a cut space, and then wipe the area clean with flux or tuner cleaner so the break is obvious.

If you must resolder a trace, scrape the green masking from both sides of the cut, wipe it clean, and flow solder on both sides, but *don't* bridge the trace with solder. Instead, take a piece of bus wire or stripped wire-wrap wire, form it into an 'L' shape, and solder the base of the L across the trace. Then cut off the excess. Bridging with solder alone is dangerous because it can look fine, but really be attached only by a glob of flux, or be attached so weakly that flexing the board will crack the solder off.

5. Piggyback on Z45 a 2102, soldering pins 1, 2, 3,
- 4, 5, 6, 7, 8, 9, 10, 13, 14, 15 and 16.
6. Piggyback on Z24 a 74LS86.
7. Piggyback on Z25 a 74LS10.
8. Piggyback on Z6 a 74C04.
9. Run a wire from Z60, pin 5 to Z25, pin 13.
10. Run a wire from Z30, pin 13 to Z25, pin 12.
11. Run a wire from Z25, pin 11 to Z45piggy, pin 11.
12. Run a wire from Z45piggy, pin 12 to Z60, pin 4.
13. Run a wire from Z45piggy, pin 12 to Z25piggy, pin 1.
14. Run a wire from Z63, pin 12 to Z25piggy, pin 13.
15. Run a wire from Z25piggy, pin 2 to Z27, pin 5.
16. Run a wire from Z25, pin 12, to Z6piggy, pin 1.
17. Connect together Z6piggy, pins 2 and 3.
18. Attach one terminal of the 100,000-ohm trimpot to Z6piggy, pin 4.
19. Attach the other terminal of the 100,000-ohm trimpot to Z6piggy, pin 5.
20. Attach one end of the 330-pf capacitor to Z6piggy, pin 5.
21. Connect together Z6piggy, pins 6 and 13.
22. Attach the other end of the 330-pf capacitor to Z6piggy, pin 12.
23. Attach one end of the 0.033-mf capacitor to Z6piggy, pin 12.
24. Attach the other end of the 0.033-mf capacitor to Z6piggy, pin 11.
25. Connect together Z6piggy, pins 9 and 10.
26. Run a wire from Z6piggy, pin 8 to Z24piggy, pin 1.
27. Connect together Z24piggy pins 1 and 4.
28. Run a wire from Z63, pin 12 to Z25piggy, pin 3.
29. Run a wire from Z42, pin 12 to Z25piggy, pin 4.
30. Connect together Z25piggy, pins 5 and 14.
31. Run a wire from Z25piggy, pin 6 to Z27, pin 4.
32. Disconnect the wires running from ZFLOP pins 8 and 9 to ZMODE.
33. Run the wire from ZFLOP, pin 8, to Z24piggy, pin 2.

34. Run the wire from ZFLOP, pin 9, to Z24piggy, pin 5.
35. Run a wire from Z24piggy, pin 3 to ZMODE, pin 1.
36. Run a wire from Z24piggy, pin 6 to ZMODE, pin 15.

The modification is complete. Using the individual reverse video is not the easiest process, but works nonetheless. Printing CHR\$(0) through CHR\$(31) results in control codes being acted upon; CHR\$(32) through CHR\$(127) now produce the full range of ASCII letters; CHR\$(128) through CHR\$(191) produce graphics characters; and CHR\$(192) through CHR\$(255) produce the 63 possible TAB positions.

To use the individual character reverse, then, you cannot PRINT the CHR\$ value. Instead, you must POKE the value onto the screen. Granted, this is a pain, but when a program is completed, the prompting can be extraordinarily effective. Besides, it's not that hard. Try this:

```
10 CLS : Y=15360
20 FOR X = 0 TO 255
30 POKE Y,X
40 Y=Y+1
50 NEXT X
60 GOTO 60
```

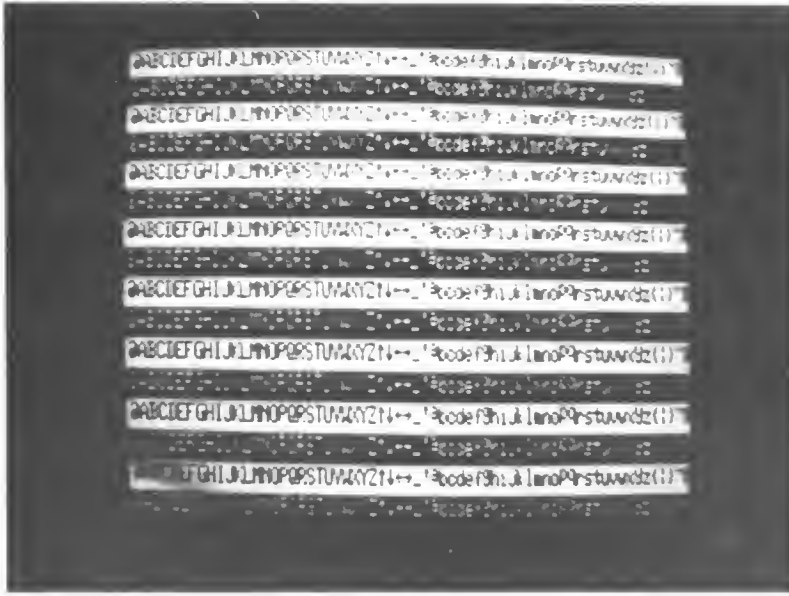
Listing 4-1. Individual character test program.

Chances are that a group of reversed characters appeared, but the reversal band didn't match up with the actual characters. That's the reason for the 74C04 and the 100,000-ohm potentiometer. Run the following test program:

```
10 CLS
20 FOR X = 16040 TO 16060
30 POKE X,255
40 NEXT
50 GOTO 50
```

Listing 4-2. Individual character alignment program.

Now adjust the potentiometer until the crosshatches are centered precisely within the white band (or rather, so the white band is in the correct background position).



Complete font of reversed characters. Because only 64 characters remained in the available set, upper and lower case were chosen as opposed to numbers and symbols.

That is a simple example of the individual character-reverse process, using BASIC. By using ordinary display routines in machine language programs, the individual character reverse is an entirely trivial matter, but really putting it to work in BASIC requires a bit of fancier footwork.

Using it through BASIC can be done without POKEing large numbers of characters in prearranged locations. Instead, the position of the cursor on the screen can be determined, and the characters POKEd in place from there. The subroutine below is useful:

```
10000 X = PEEK(16416) + 256*PEEK(16417) : RETURN
```

This subroutine determines the position of the cursor and assigns it to X. Then characters may be POKEd as follows:

```
40 GOSUB 10000
50 A$ = "THESE ARE REVERSE LETTERS"
60 FOR N = 1 TO 25
70 Q = ASC (MID$ (A$,N,1))
80 POKE X,Q
90 NEXT
100 GOSUB 20000
```

Listing 4-3. Individual reverse demo program.

The GOSUB 20000 executes the following one-line subroutine:

```
20000 ZH = FIX (X/256) : ZL = X-ZH*256 :
      POKE 16416,ZL : POKE 16417,ZH : RETURN
```

This short routine restores the cursor position after the POKEs have taken place. There are two very important things to note:

1. If the POKE is to take place on the last line of the screen, make absolutely certain that the value of X does not exceed 16383, because this will POKE nasty values into BASIC vectors beginning at 16384. A test for X greater than 16383 can be made so:

```
10005 IF X>16383 THEN X=96383 : GOSUB 20000
      : PRINT : RETURN
```

This will have the effect of restoring the cursor, printing a carriage return, and finishing the text to be printed.

2. The POKE feature does not include a carriage return, so this method of printing a reversed message has the effect of a PRINT; (PRINT semicolon) statement. Immediately follow the return-from-subroutine with a PRINT statement if the rest of the message is to be printed on the next line.

A few peculiarities may arise with this modification; among them:

1. The 'fill' character of Electric Pencil may change. This is because any program defining graphics characters as 192 to 255 instead of 128 to 191 is not using them according to the original Radio Shack specifications.
2. Single reversed letters scattered throughout text may not work. This is due to the extremely fast requirements of the circuit, and the fact that certain integrated circuits may not be up to it. Two characters together, however, will print properly.
3. The timing is so crucial that temperature may offset the image slightly. Use polystyrene or polycarbonate capacitors, never ceramic disc capacitors, for this modification.
4. With a screen full of reversed letters, some increase of the brightness control may be needed.
5. The full character set is not available because more hardware would be necessary to obtain the logic information necessary to select out, for example, letters and numbers or letters and symbols.

Although it sounds like I am doing a lot of warning about the limitations of this sort of modification, you should realize that the TRS-80

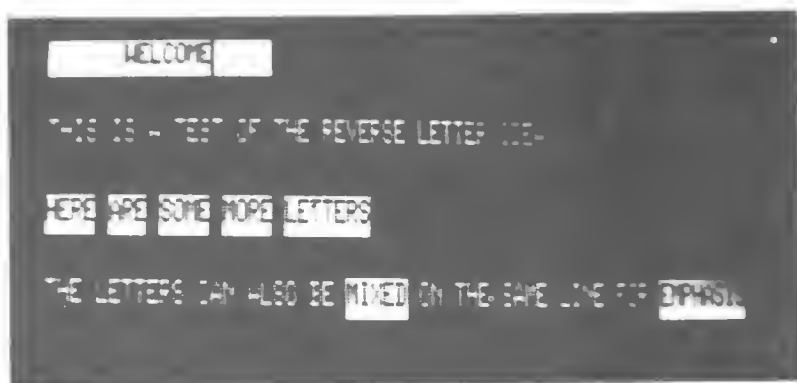


Photo 4-13. Individual reverse screen example.

Reverse lettering provides an emphasis; flashing from one to the other is extremely effective for prompts.

was never meant to do this sort of thing, and the fact that it does work is remarkable. Once it is in place, you will wonder how you could have created reasonable self-prompting programs without it. See Photo 4-13 for proof of the impressive results.

A complete circuit containing the reverse video, upper/lower case, and individual reverse video modifications appears in Figure 4-10, below:

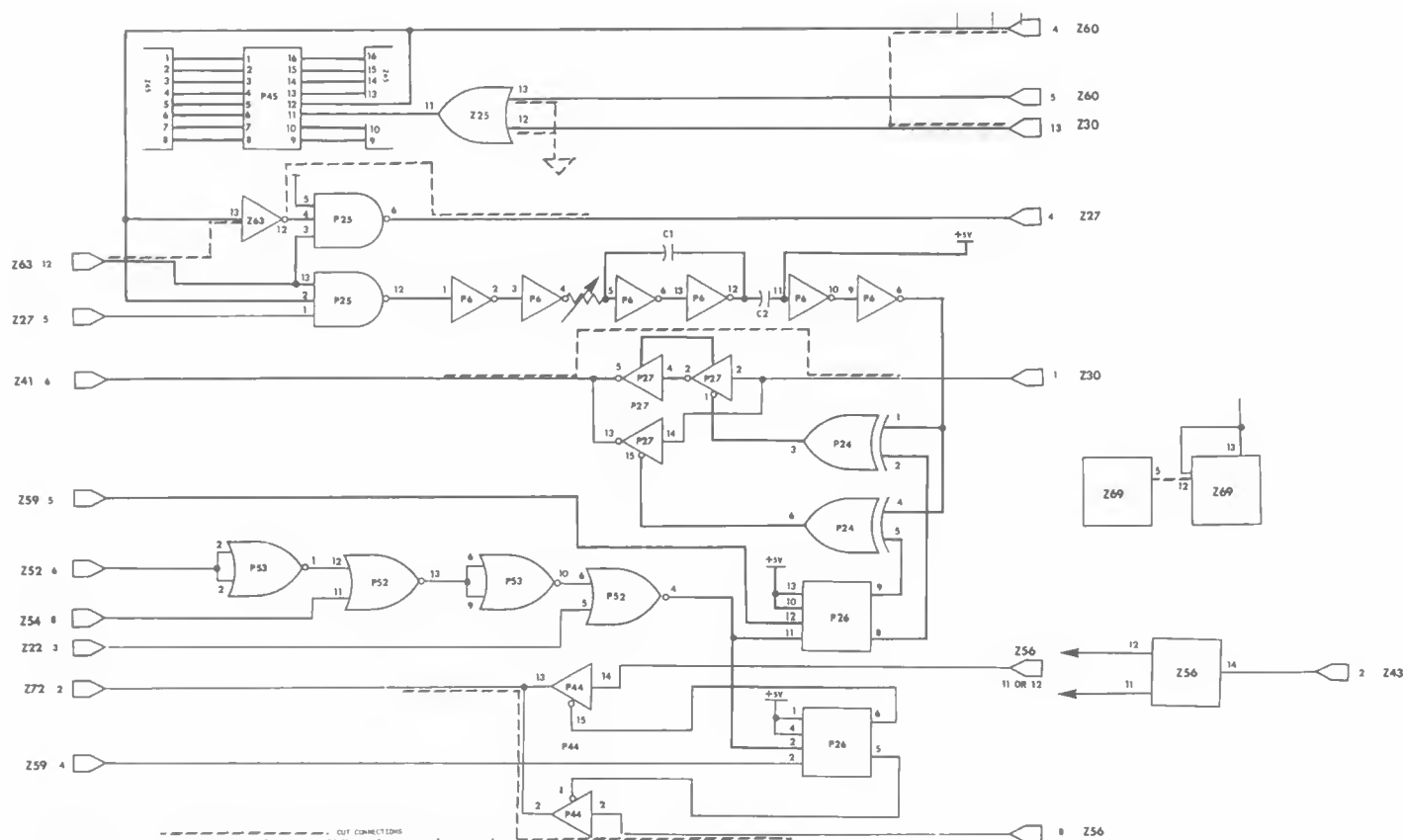


Figure 4-10. Complete video modifications schematic.

## Stepping on the Accelerator

When you first unpacked your TRS-80 and tried a few calculations and displays, you were likely amazed at the speed with which the TRS-80 was able to respond. But certainly as you used the machine, you realized that it could spend quite a while performing complex tasks. It was then that you joined the ranks of the dozens of thousands of micro users searching for a faster running computer.

When the TRS-80 was designed, the options for higher speed were put in place. Normal and 200 percent speed options were right there on the circuit board, and 150 percent speed required the routing of but a single circuit trace; yet the final design opted for the lowest speed (1.77 MHz).

This modification releases the permanent connection of the 100 percent (1.77 MHz) clock and allows the computer to switch back and forth between this clock rate and a rate one-and-one-half times faster – 2.66 MHz – or two times faster – 3.55 MHz. There are a few pitfalls, but the actual modification is a simple one. For it, you will need the decoded port 254 explained in the reverse video section of this chapter (74LS02), the other half of the flip-flop (74LS74), and one 74LS367.

Before you perform this modification, there are a few things you should know. For a reason which I have yet to determine, some Level II machines with the *two-chip* ROM set will not accept a speed-up successfully, though most will. Second, later units have been manufactured (or earlier units may have been retrofitted) with a small board known as 'XRX III', a synchronous, 500-baud wave shaper. Cassette load at the 750- and 1000-baud rates will not work unless the modification is deactivated (for details on the cassette system, see the Supplements to Chapters 3 and 6).

The third item concerns the expansion interfaces manufactured since January 1980. Unfortunately, these interfaces must also be modified to accept a higher-speed CPU. But this modification (see Chapter 5) is quite simple and very reliable. Finally there is the question of memory speed. If your TRS-80 is *very* early and the 16K RAMs were shipped with the unit, there is a small chance that one or more of these RAMs will not be capable of running at 3.54 MHz — and an even smaller chance of not operating at 2.66 MHz.

That said, I'll turn to the modification itself. Locate Z56 on the circuit board. Cut the foil trace that leads from this pin to the hole that is plated through the circuit board. Mark that hole for future reference. By cutting this trace, you separate the 1.77 MHz output of Z56 from the clock input of the Z80 processor.

If you have not created the port 254 decoding used in the reverse video modification, you will need to piggyback a 74LS02 on Z53. Bend all the leads except 7 and 14 parallel with the body of the IC, and seat it atop Z53 with the notch or dot pointing in the same direction as the rest of the integrated circuits on the board. Solder pins 7 and 14 to Z53; this piggybacked circuit will be called ZPORT.

Locate Z52; run and solder a wire from pin 6 to pins 2 and 3 of ZPORT. Find Z54. Run and solder a wire from pin 8 of Z54 to pin 11 of ZPORT. Pins 1 and 12 of ZPORT are connected together. This completes the decoding of port 254 (hex FE). To add the necessary OUT signal, run a wire from Z22 pin 3 to ZPORT pin 5. Solder together pins 8, 9, and 13 of ZPORT; solder together pins 6 and 10 of ZPORT. The BASIC command OUT 254,X will activate the signal found at ZPORT pin 4.

Also a part of the reverse video modification was the piggybacking of a 74LS74 atop Z26. Bend all leads parallel to the body of the IC

except power pins 7 and 14. With the integrated circuit oriented in the same direction as Z26, mount it there, soldering pins 7 and 14 to the IC below. This IC is called ZFLOP.

Run a wire from the decoded signal at pin 4 of ZPORT to pin 3 of ZFLOP. Z59 has a data line (bit 0) at its pin 4; run a wire from that pin to pin 2 of ZFLOP. Now run wires connecting together pins 1, 4 and 14 of ZFLOP. With these connections made, OUT 254,0 will flip the circuit, and out 254,1 will flop it. Note that OUT 254,0 and OUT 254,2 were the flip-flop commands for the reverse video.

Now the 74LS367 is put in place. As with the other ICs, bend all leads except the power pins (pins 8 and 16 on this circuit) parallel with the body. Place the 74LS367 on Z44, and solder pins 8 and 16 to it. Call this ZFAST.

Locate pin 8 of Z56, whose trace is already cut. Run and solder a wire from this pin to pin 2 of ZFAST. Run a wire from pin 3 of ZFAST to the plated-through hole, which was previously marked for reference. Take care that this is the correct hole before soldering; it is the other end of the trace cut from Z56.

The normal clock will now be reunited with the CPU. Run a wire from pin 5 of ZFLOP to pin 1 of ZFAST and solder. When OUT 254,1 is executed, the normal speed will flip in place. The other speed output buffer is wired now. Tie pins 13 and 3 of ZFAST together. Last, run a wire from pin 6 of ZFLOP to the second gated section of ZFAST, pin 15.

The higher speed will now be selected. The options are two, and you may try either 150 percent normal speed or a hot 200 percent normal speed.

*150 percent normal speed:* Locate Z43, pin 2, which is the 5.32 MHz clock normally used in the video divider chain. Run a wire from this pin 2 to Z56 pin 14. That pin is the input of an unused divide-by-two segment of Z56. The output (2.66 MHz) of this divider is present at Z56 pin 12. Run a wire from there to pin 14 of ZFAST.

*200 percent normal speed:* Locate Z56, pin 11. This is the 3.54 MHz clock not used in the TRS-80. Run a wire from this pin 14 of ZFAST. For this super-fast mod, the memory select circuits must also be dealt with. Locate Z69, and cut the trace running from pin 5 to pin 12. Connect pin 12 to pin 13. This speeds the memory-select process (from MREQ and RD) just a tad, but enough to cope with the 200 percent modification.

## Level I and Level II Together

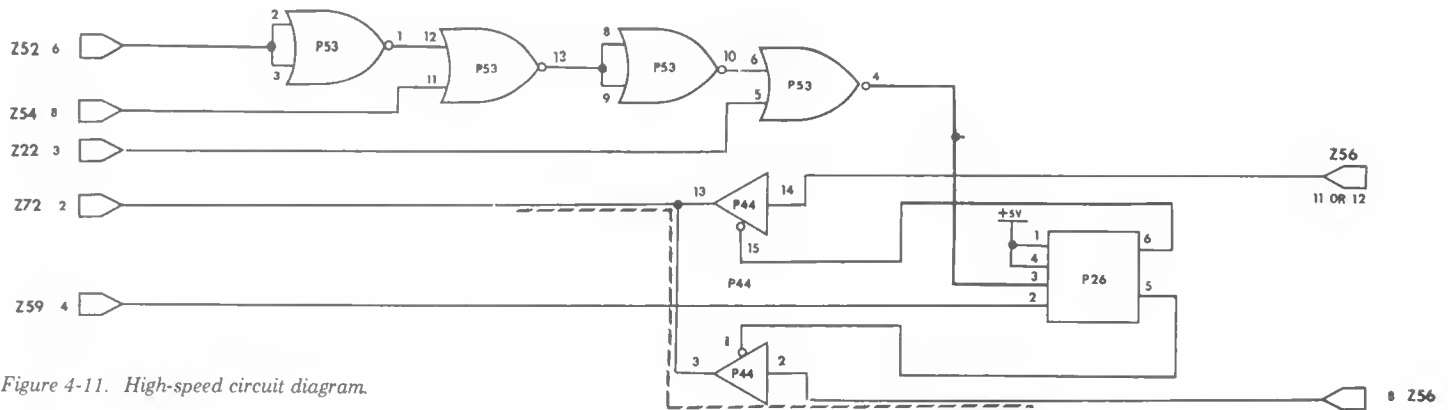


Figure 4-11. High-speed circuit diagram.

The high speed modifications are now complete. Reassemble the TRS-80 and run a test by trying OUT 254,0 and OUT 254,1, which will flip and flop the speeds. The following short BASIC program will give a good demonstration of the speed differences:

```
10 FOR B = 0 TO 1
20 CLS
30 OUT 254,B
40 FOR X = 1 TO 50
50 PRINT X;
60 NEXT : NEXT
70 GOTO 10
```

Listing 4-4. High-speed demo/test program.

In addition to the modifications themselves, it is often useful to know which speed is active. There are ways of telling after working with the higher speeds: the very slight herringbone in the video monitor (turn contrast high to see it) changes pattern, programs work faster, etc. However, a bipolar LED is ideal for this.

To pin 1 of ZFAST, attach one end of a bipolar LED. Attach the other pin to a 470-ohm resistor, and run this resistor to pin 15 of ZFAST. The light will show green for one speed and red for the other; by reversing the center pin of the LED, the color pattern selected can be reversed. I use red for the normal speed, and green for the higher speed.

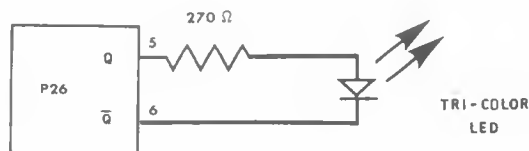


Figure 4-12. LED high-speed indicator circuit diagram.

## Level I and Level II Together

Little defense of Level I BASIC has been made, but from my point of view, it's a delightful, compact little BASIC. My own familiarity with it is relatively recent, because I ordered my TRS-80 with Level II installed. Aside from the maddening lack of key rollover, Level I seems an ideal teaching language, especially for youngsters. Together with the excellent Level I manual created by David Lien, it is a fine introduction to the language, and to computers themselves.

Enough for the defense. The problem with Level I is that it is not Level II, and the bulk of we TRS users have Level II BASIC installed. Level I can nonetheless be co-resident, and there are three ways to do it: install the Level I ROM with a switch (this is the method presented here); use a disk system with the Level I-in-Level II program (offered by Apparat and others); or relocate and rewrite Level I a bit and burn it into an EPROM placed in high memory.

Level I ROMs are getting harder to obtain because of the Model I's discontinuance, but cooperative repair centers or franchise Radio Shacks ('Associate Stores', Tandy calls them) can often provide the ROM for a few dollars. Although I am not one to encourage software copying, I do feel that as a TRS-80 purchaser, you paid for your Level I ROM if you bought the unit with one. If it was not returned when Level II was installed, then Radio Shack owes you one. Try to get it.

The other way is to order the ROM as a replacement part; Radio Shack's latest replacement number is MX4126, and they charge \$71.25 (!) for it. Finally, you can borrow one from someone who has it and copy it into a 2732 EPROM, but that's still about a \$40 investment.

Let's assume you're able to obtain a Level I ROM. The ones you want are marked National, and are identified with parts numbers M2316E/MMS258ET R/N and S/N, or Motorola, marked 7807 and 7804 or 7831-BASIC1-ROM A and 7832-BASIC1-ROM B. Ideal is the single-chip ROM from Motorola or Rockwell, marked 7809 and 7845, respectively (phew!). Photo 4-(?) shows the two-chip (7831/32) set and the Rockwell (7845), mounted on an aluminum-foil-covered vegetable tray.

You do *not* want the chips marked Intel; these are 2716 EPROMs, and the wiring is complex. About the only thing they are good for is scraping off the label and erasing under ultraviolet light. Then you have two spanking new \$12 EPROMs.

Check next to see if you have the two- or three-chip Level II ROM set. If you have the three-chip set, there will be a connector cable running to a separate board taped to the main circuit card. If not, both ROM sockets will be filled, and installing Level I will be more difficult. Last of all, make sure your circuit board is a 'D' board, 'G' board or later. 'A' boards won't do. (The number is part of the lettering silkscreened on the board, such as 1700069D or 1700069G).

First will be instructions for installing the ROM in the TRS-80's with outboard Level II ROMs. Mount a double-pole, double-throw switch conveniently, but discreetly enough that you won't be knocking it into Level I in the middle of a four-hour data sort.

Open up the TRS-80, and note where the Level II ROM cable is plugged. Four (or six) other wires run from this Level II board to the rest of the circuit card. Find these locations:

1. The green wire on the Level II board, connected near the underside of dip shunt X3.
2. ROM socket Z33 or Z34 (whichever is empty), pins 18 and 20.

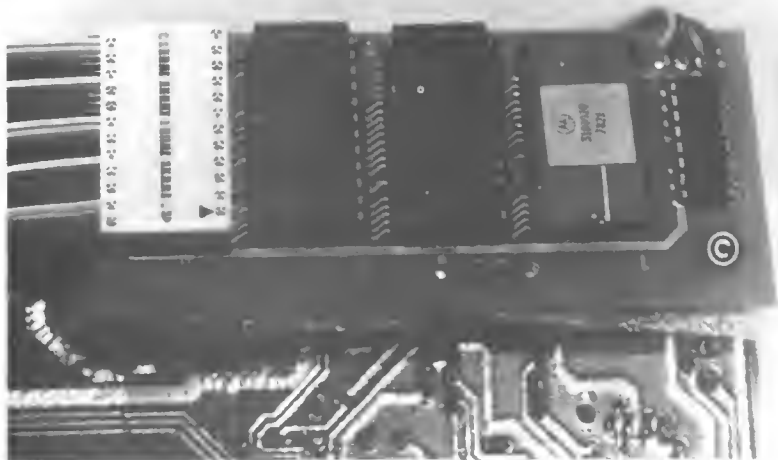
Cut the traces leading from pins 18 and 20 of the unused ROM socket. Add a short length of wire between pin 20 and the far end of the trace that used to lead from pin 18. Solder a long white wire to pin 18. Remove the far end of the Level II board's green wire from its connection point near shunt X3, and solder a red wire there. Solder a blue wire to the 5-volt supply found at Z57, pin 14. Using Figure 4-13, run the white, green, red and blue wires to the double-pole, double-throw switch.

Add two 1000-ohm resistors to the circuit to hold the ROM chips inactive when they are not in use. Without them, inadvertent selection might take place, and the running program (in either language) might crash.

If you've got the single-chip Level I ROM, you're all set to go. If not, the fun begins here. Locate the notch on each Level I ROM chip, and line up the two chips with each other, *precisely pin for pin*. Now piggyback one atop the other and solder all 24 pins so that the result is a single, hulking, integrated circuit. Solder carefully, keeping the bottom IC anchored in conductive foil. The foil will act as a static remover and heat sink, both essential in this mod.

Blobs of solder can be removed with some sort of solder-wick or a solder-removing vacuum tool. This entire chip is then inserted in the empty socket, and the unit is ready for testing. Level II gives you the expected MEMORY SIZE?, while Level I only reports 'READY'.

If you have the two-chip *Level II* ROM set, there's a bit more work to do. One ROM must be removed and piggybacked on the other, except for pin 20. Bend pin 20 out straight on ROM A (Z33), and piggyback it on ROM B (Z34). Run a wire from ROM A pin 20 to Z74 pin 9. Bend pin 20 out straight on ROM B (Z34), and run a wire from this pin to Z74 pin 12. Now insert your *Level I* ROMs in the socket for Z33.



Level II ROMs: 3-chip set on individual board at the end of a cable which is plugged into the main CPU card.

## Level I and Level II Together

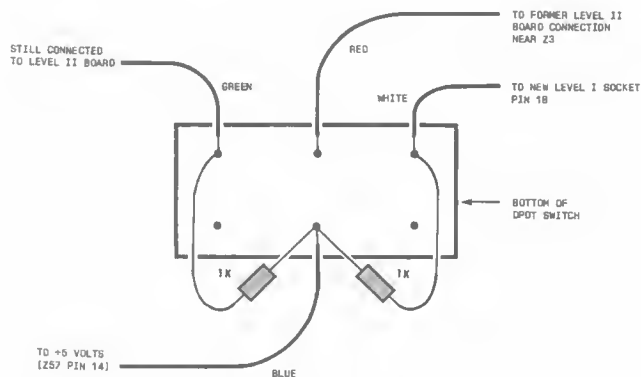


Figure 4-13. Level I and II switch wiring (3-chip set).

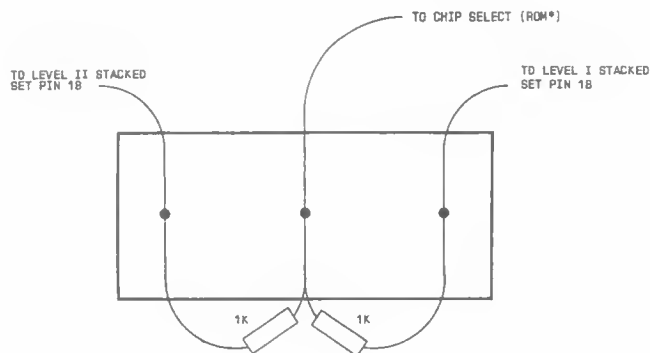
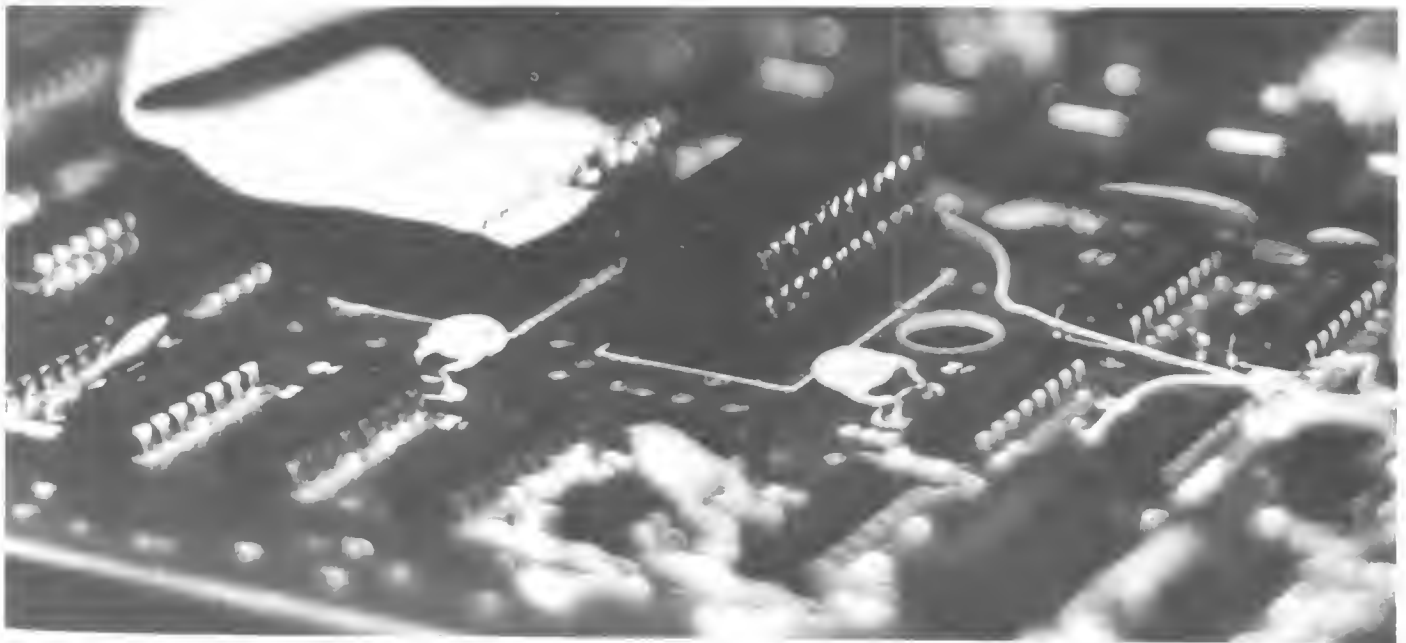


Figure 4-14. Level I and II switch wiring (2-chip set).

Since these languages use memory and pointers in such a different manner (Level I is not by Microsoft, as I understand it), they cannot be switched from one to the other directly. You must power up to the language you want.

This can present a problem. If you have installed various speed and video modifications, you may get the mode you don't want. Level I, lacking OUT statements, can't recover from this. To avoid it, solder a wire to Z53, pin 12, and attach it to one end of a pushbutton; to the other end of the button, solder a wire to ground (Z53 pin 7 is fine for that). This is the system reset (SYSRES) signal, a true restart to 0000.

To use it, power up to Level II, get in the video/speed mode you wish using OUT statements, switch to Level I (the screen will likely display a mess of @9@9@9's), and press the SYSRES button. A READY should appear. Now dig back in the Level I manual to determine if the amount of memory reads correctly.



*Two-chip Level I ROM set can be added by piggybacking them and soldering them together. Together with the snaking Level II cable, they form a surrealistic electronic landscape.*

## On Relocatable Code

A great noise is often made by programmers and users of the TRS-80. The clamor is for an elusive programming quality which allows code to be placed anywhere in memory and to function from that place. The cry is for relocatable code, but it is often called for without understanding either the process or the sacrifices it requires. In this section I will take a look at the various levels of relocatability, and the ways it can be applied in low-level languages.

The most 'relocatable' code TRS-80 users see is BASIC itself. No matter what configuration changes the system undergoes, whether they be the use of low-memory utilities, DOS, etc., a normal BASIC program will load and run. A BASIC program's portability depends on the premise that it is *interpreted*, containing statements that are equally meaningful irrespective of the program's position in memory. Occasionally that position gains importance, as when numerical variables, strings or arrays are referenced within program lines, but normally those positions do not become significant until the program is in operation.

Unless the BASIC program is truly a hybrid (having other, lower-level languages artificially embedded in the code), it is eminently portable. All references are resolved by its interpreter, and relocatability of BASIC is assumed in the very nature of it and similar high-level, interpreted languages. In fact, that is the nature of a real 'language' as we know it as opposed to the mysteries of 'code'.

The distinction between language and code is not an accepted one, and the differences will be characterized only for descriptive clarity within the context of this article. By 'language', I intend BASIC, FORTRAN, Pascal, and others, as well as assembly language. By 'code', I suggest machine code as written in binary or hexadecimal for direct insertion in memory. The assembly language/machine code interface is curious and I will address it in terms of the conflict between understanding and using a language as opposed to a code.

## At the High Level

The highest levels of languages are the so-called 'descriptive' languages, like IBM's report-generator, RPG II. A programmer need only describe a group of tasks and the order in which they should be performed, and they will be magically executed. Full-dress accounting

reports, billings, and record-keeping can be done with a language like RPG II. Relocatability is at best the task of a dour 'systems engineer'.

Likewise, BASIC normally requires no thought to its arrangement beyond the proper sequence of line numbers. Statements are arranged in the programmer's chosen order, and the interpreter selects each one for execution in that order. Line numbers are the programmer's relocatable reference. Languages containing no line numbers often use an implicit 'top-down' order; that is, once a task is completed, it is never returned to. Still, the program is created with an arbitrary arrangement selected by the programmer and evaluated by an interpreter before its execution.

Languages which use compilers rather than interpreters work in much the same manner. Whereas an interpreter selects and executes each command and operand during the program's run, a compiler produces a block of machine-coded information from high-level commands and operands. The process is completed before executing the program, so valuable space and running time can be saved. In either case, arbitrary, high-level language can be moved about, deleted, inserted, or otherwise altered with little regard for the program's ultimate position within the bowels of the computer.

In fact, the question of relocatability is implicit in the term 'high-level language', and is an essential of its high-level quality.

The reason the question arises at all is because, when writing machine-coded utilities, many programmers feel that emulating the portability of a high-level language is crucial. It becomes even more desirable when programming is done for a user rather than another programmer, a user whose contact with the machine may go little further than that of a traditional computer 'operator'.

Accomplishing relocation can be an easier task on the Model I than on other computers simply because of the extensive ROM-based operating system. Before examining some ROM uses (and its advantages and pitfalls), let's have a look at the qualities and constraints of relocatable machine code.

As a user interested in accomplishing a task, I might need to select from a group of utilities including sort routines, multiple-precision transcendental functions, mundane key-debounce and upper/lower case drivers, and so forth. Ideally, the order in which I select and load these routines should be irrelevant to their

operation, in much the same way that utilities can be 'ordered up' on a mainframe.

Microcomputer software authors have not generally respected this need; in some measure that is forgivable because the approach and intent of different utilities from a variety of vendors demonstrates that relocatability is only one of many potential conflicts to be resolved.

### Hunting Through Memory

Nearly any well-written program is tailored to meet certain requirements which may not easily co-exist. Among them are:

**Speed of execution.** This applies in two ways. The first is the speed at which the *need* for the utility is evaluated - an example might be a routine scanning a keyboard for a unique combination of characters. This should be accomplished expeditiously, since no real program 'work' is being done. The second speed consideration is that, once called for, the main routine should complete its work handily.

**Conservation of memory.** In a system of unlimited size, speed can be optimized by replication of routines rather than use of subroutines. In appliance-level microcomputers like the TRS, however, support utilities cannot be allowed to consume memory space needed by the operation of the target program.

#### Resolution of Conflicts.

All operating systems make use of areas of critical reserved RAM, which are employed to tie together the major pieces of hardware and blocks of software. These patch points (I prefer this term to 'vector', which also means a disease-carrying insect) may be needed by many other utilities as well as the operating system, and must be redirected carefully.

**ROMability.** The traditional approach states that programs must be created so that they can be committed to ROM at any time. That is, they should neither modify themselves nor contain variable data within their bounds. This approach can certainly be argued against (I shall).

Three of these four elements - speed, size, and memory sharing - are normally considered in utility programming, and become critical in relocatable programs.

In addition, factors to be considered exclusively for program relocation are:

**The placement of the program in memory.** This can be a single- or multiple-step process. For example, during loading from disk or tape, a program may temporarily reside in a single block of memory, only to separate itself into smaller blocks which are shepherded to other memory fields. *Electric Pencil* is one such program.

**Absolute jumps within a program.** The program counter is instructed to take on a new value, a value specified as an address. These jumps are, when taken, faster than the relative branches, because the program counter need not calculate an offset to its current address.

(indent absorber) **Subroutine calls within a program.** Effective use of memory space makes subroutines very attractive, but they require that the program counter be assigned a specific new value for their duration.

**A relocation block.** If the program is provided to a user in object-code (SYSTEM) format, some portion of the program must be dedicated to relocating itself in another area of memory.

### Assembly Programming

When programming for programmers, the most convenient way to effect relocatable code is to use assembly language, and supply that to the user. By labeling all jumps and calls, such a program becomes 'relocatable', somewhat in the sense that a program conceived in a high-level language is relocatable. In fact, in its assembly phase, machine coding is a high-level language.

21003C	8000	ORG	8000
11003C	8000	LD	HL,3C00H
01FF03	8003	LD	DE,3C01H
3620	8006	LD	BC,3FFH
E080	8009	LD	(HL),20H
21E942	8008	LDIR	
7E	8000	LD	HL,42E9H
A7	8010	LD	A,(HL)
C29719	8011	AND	A
C30043	8012	JP	NZ,1997H
	8015	JP	4300H
	4300	ORG	4300H
C08843	4300	CALL	4388H
23	4303	INC	HL

Listing 4-5. Machine coding in assembly language.

Above is a section of assembly programming. It makes several assumptions, some valid and some not. Let's consider that this program has cleared the screen (which it would in the TRS-80

configuration) after having performed some operations on a BASIC program. It checks location 42E9, the start of BASIC, for a zero. If it does not find a zero, it assumes a syntax error. Otherwise, it moves to location 4300, where it immediately calls a subroutine at 4388.

This is perfectly valid code, but here's a look at another way of creating it in this high level language:

```

04XA FIGURE 2      SETUP EQU 8000H
                     VIDEO EQU 3C00H
                     SCREEN EQU 03FFH
                     BASIC EQU 42E9H
                     PROGRAM EQU 4300H
                     SYNERR EQU 1997H
                     TESTER EQU 4388H
                     ORG $
21003C 8000          SETUP LO HL,VIDEO
11013C 8003          LD DE,VIDEO+1
01FF03 8006          LD 9C,SCREEN
3620 8009            LD (HL),20H
ED80 8008            LDIR
21E942 8000          LD HL,BASIC
7E 8010              LD A,(HL)
A7 8011              AND A
C29719 8012          JP NZ,SYNERR
C30043 8015          JP PROGRAM
                     ORG $
C08843 4300          CALL TESTER
23 4303             PROGRAM INC HL

```

Listing 4-6. Machine coding in assembly language.

This assembly program, using its high-level, interpretive capabilities, can be easily relocated by a programmer changing a few equates. But it is not relocatable because it does not meet the criteria outlined above – i.e., it contains a specific origin (or rather, two), absolute jumps, calls to subroutine, and has no visible relocation block. Fortunately the start of BASIC has been defined in the table of equates, for it too can be a variable element where low-memory alterations have been made.

As a programmer's utility, this method is acceptable. As a public program, it is questionable. The solutions, however, are neither easy nor obvious.

## Relocation Blocks, Time, and Space

The first option is to provide a relocation block. The amount of coding the programmer must perform is increased, but memory use is not affected because this relocation block can be deleted upon relocation. A relocation block is provided with Radio Shack's KBFIX, for example, and with other utilities. The sample above might have a relocation block which asks for a new base address in protected memory, protects that memory, adjusts its internal addresses, and automatically checks for the beginning of BASIC program storage and other patch points it may require.

For very short utilities, the relocation block may be longer than the utility itself, because it demands a user prompt and input, minimum error checking, calculation of base address, and adjustment of patch points. Even using ROM routines (such as 28A7 to display a message, 1BB3 to accept user input, 1E4A for numeric conversions), this process is lengthy.

One of the most painful aspects of this relocation is the adjustment of internal addresses; absolute jumps, calls, and table locations are among the addresses requiring such changes. Then what of the questions of speed and conservation of memory? Programs that move themselves during the course of operation must carry the excess baggage of a lengthy self-relocation block or use a plethora of relative instructions which can cost time during program execution.

Let's address these speed and space questions. Table I lists a number of the relative instructions in the Z-80 set, along with the bytes, machine cycles and time required for their execution. In contrast, Table II presents similar absolute Z-80 instructions, their execution speeds, and the percentage of time and space they require as opposed to the relative instructions.

Z-80 INSTRUCTION	BYTES	T-STATES	TIME AT 1.77 MHz
LD r, (IX+d)	3	19	10.7 uS
LD (IX+d), r	3	19	10.7 uS
LD (IX+d), n	4	19	10.7 uS
ADD A, (IX+d)	3	19	10.7 uS
INC (IX+d)	3	23	13.0 uS
RLC (IX+d)	4	23	13.0 uS
BIT b, (IX+d)	4	20	11.3 uS
SET b, (IX+d)	4	23	13.0 uS
JR e*	2	12	6.8 uS
OUNZ e	2	13	7.3 uS

When this instruction is conditional, and the condition is not met, it uses 7 T-states (3.9 uS).

### Note:

'r' = registers A, B, C, D, E, H or L.

'd' = displacement byte 00 to FF

'n' = single-byte integer 00 to FF

'b' = bit position 0 to 7

'e' = offset byte 00 to FF

Z-80 INSTRUCTION	BYTES	T-STATES	TIME AT 1.77 MHz	% TIME & SPACE VS. REL. INSTR.
LD r, (HL)	1	7	3.9 uS	37% 33%
LD (HL), r	1	7	3.9 uS	37% 33%
LD (HL), n	2	10	5.6 uS	53% 50%
ADD A, (HL)	1	7	3.9 uS	37% 33%
INC (HL)	1	11	6.2 uS	48% 33%
RLC (HL)	2	15	8.5 uS	65% 50%
BIT b, (HL)	2	12	6.8 uS	60% 50%
SET b, (HL)	2	15	8.5 uS	65% 50%
JP nn*	3	10	5.6 uS	83% 150%
JP NZ, nn	3	10	5.6 uS	77% 150%

When this instruction is conditional, and the condition is not met, it still uses 10 T-states, 3 more than the relative branch instruction. In programs where such a condition is generally not met, the JR e instruction will save both time and memory.

Note:

'nn' = two-byte integer 0000 to FFFF

Relative instructions in many cases command considerably more time than absolute ones. How does this reflect on program speed? The design engineers for the 6809 chip reported in *BYTE* a survey of 6800 instruction class usage based on static analysis (i.e., with the program not running) of 25,000 lines of source code. Because of the considerable differences in chip architecture between the Z-80 and the 6800, especially in regard to the Z-80's multitude of registers, this information (see Table III) is not directly applicable. Yet it is instructive, particularly in the percentage of subroutine calls and branches, instructions similar in the two microprocessors.

TABLE III	
Use of 6800 Instruction Types	
Loads (movement from register to register, and from memory to a register)	23.4%
Stores (movement from a register to memory)	15.3%
Calls and returns (absolute addressing)	13.0%
Conditional branches (relative)	11.0%
Unconditional branches (relative) and jumps (absolute)	6.5%
Others	30.8%

(Adapted from *BYTE*, 4:1, January 1979, p. 26)

A remarkable 30 percent of program space is dedicated to motion of the program counter. This might suggest that a program using absolute instructions would be 15 percent faster overall than one using relative branches and calls. Since relative calls and returns are complex, however (see below), the time savings might be even greater. Absolute jumps use more space than relative ones, so space savings, though significant, would not be as impressive.

What are the practical implications of this speed differential? A 1,000-byte subroutine (long by microprocessor standards) with no internal loops might execute in four milliseconds, as opposed to five milliseconds for such a routine written exclusively with relative functions. On the average, it might be 200 bytes longer. This means 200 iterations per second as opposed to 250. For a complex mathematical function which iterates through a routine many times, it could mean the difference between a four-hour and a five-hour run. For a patch to a

keyboard scan, it could mean adding only a few seconds to an hour's program time. And for routines involved in occasional string handling, printing, low-speed input/output, error reporting, or a host of other applications, the loss of time is insignificant.

The application itself is the answer to the viability of relative coding. In time-sensitive applications (music-generation routines, for example), and where every byte of code is crucial, relative instructions can interfere with a program's effectiveness. In the bulk of TRS-80 applications, which are seldom so time-conscious, relocatable coding through use of relative instructions can be very successful.

## Creating Relocatable Code

**The Relative Branch.** The first coding choice is the two-byte instruction for changing the program counter. No matter where the program counter is currently pointing, it can be shifted by specifying a relative branch. When the high bit of the specified offset is zero, the program counter is moved ahead (for example, 18 37 adds 37 to the program counter). When the high bit of the offset is one, the program counter is moved back, in effect subtracting 80 before adding the offset:

5236	5236	:	5236	5236
+ 00	+ 80	:	+ 7F	+ FF
5236	5186	:	5285	5236
(5236+0)	(5236-100+80)	:	(5236+7F)	(5236-100+FF)

Especially gratifying is the notion that the page of the address (the high byte) is incremented or decremented automatically by this instruction:

PC = 52C5	(page 52)
Execute 18 37	
PC = 52C7+37 = 52FE	(page 52)
Execute 18 37	
PC = 52FE+37 = 5335	(page 53)
Execute 18 A7	
PC = 5335+CA = 52FF	(page 52)

The branching can be done unconditionally, or on the basis of the zero or carry flags (JR, JR Z, JR NZ, JR C, JR NC). This does not provide the options available when using the absolute jump (which can act on the status of the parity and sign flags), and thus presents additional coding demands.

A special relative branch instruction, DJNZ e, automatically decrements the B register, and performs a branch to the indicated offset whenever B is not decremented to zero. Although this instruction can have many uses, its two-byte brevity makes it most attractive when executing loops.

The problem of branches outside the range of the +7F/-80 byte offset can be solved by inserting 'stepping stones' in the main program flow. At the end of any logical program section, two unconditional branches may be inserted: the first branch serves to skip past the second branch to prevent disturbing the program flow in that area. The second serves as a stepping stone for some branch which is too far (more than 127 bytes) from its eventual destination. The program can be ordered judiciously so as to avoid overusing such memory-gobbling stepping stones.

**Indexed Addressing.** One of the best features of the 6500 and 6800-series CPUs is their ability to address and manipulate memory without continually reloading a register or using an absolute address in an instruction's operand. In fact, using these processors would be incredibly cumbersome without this mode since they contain merely two storage registers (X and Y) as opposed to the Z-80's sixteen (B, C, D, E, H, L, B', C', D', E', H', L', R, I, IX, IY).

The value of the indexing features was recognized by the Z-80's designers, who added them to the 8080's primitive instruction set. As with relative branch instructions, a single-byte offset may be added to the value specified by registers IX or IY in order to produce the address of the desired memory location:

	IX = 6AA1 + 01	IX = 6AA1 + 7F	IX = 6AA1 + A1
resulting address:	6AA2	6B20	6A42

The address of the index registers themselves does not change; the resultant value is stored in temporary internal Z-80 registers while an instruction is being executed. For example, when IX = 6AA1 and the instruction –

INC [IX + A1]

– is specified, the CPU temporarily creates the resulting address 6A42 and increments its contents. The value 6A42 itself is then discarded.

When a desired value is outside the range of an indexed register, the instruction ADD IX,pp comes to the rescue. The designation 'pp' is for any of registers BC, DE, IX or SP, which can be assigned specific values representing a full 16-bit offset, rather than the 8-bit offset of the indexing itself. For example:

```

IX = 57EB
DE = 41CC
Execute ADD IX,DE
IX = 99B7
DE = 41CC

```

Since it is an offset rather than a specific address, and since it remains unchanged after the execution of ADD IX,pp, it becomes a perfect candidate for use in relocatable code.

Instructions using the index registers can be particularly valuable when lookup tables must be accessed again and again. With these commands, any registers used as pointers may remain unaltered; they need not be redefined each time the lookup routine is used. An excellent application of the index registers for five-character string comparisons is found in William Barden's *Z-80 Microcomputer Handbook*.

**Relative Subroutines.** No, there aren't any secret relative call-and-return instructions that Zilog never told us about. Rather, there is a way to create the effect of a relocatable call by using an index register in combination with relative branches.

This method was detailed in my article, 'Relative Subroutines for the Z-80', *BYTE*, 4:12, December 1979. It is a bit cumbersome at first, and makes it impossible to produce 'ROMable' code. In summary, here is how it works (see also Table IV):

1. An index register is set to some point in the program. This becomes a reference point.
2. The call is prepared by determining the offset from the *end* of the subroutine *back* to the main program flow.
3. During program execution, the offset is assigned, (using the indexed instruction LD (IX+d), n) as the *operand* of a relative branch placed at the *end* of the subroutine.
4. A relative branch is executed to the beginning of the subroutine.
5. When the subroutine reaches the end of its execution, it moves back to the program in progress.

TABLE IV			
Relative Subroutine Flow			
7000	0021B070	LD IX,70B0	;IN PROGRAM AREA
7004	.....< MAIN PROGRAM HERE >.....		
7006	.....< PROGRAM CONTINUES >.....		
.			
7040	00360B8F	LD (IX+B),BF	;RETURN OFFSET
7044	1B20	JR \$+22	;JUMP TO SUBR.
7046	.....< RESUME PROGRAM HERE >.....		
7047	.....< PROGRAM CONTINUES >.....		
.			
7066	.....< SUBROUTINE IS HERE >.....		
7067	.....< SUBROUTINE CONTINUES >.....		
.			
70B7	1B80	JR \$-41	; "RETURN" INSTR.
< NOTE THAT "B0" IN OPERAND ABOVE WAS PUT IN PLACE BY CALLING PROGRAM AT 7040. >			

This method requires careful program preparation in that the beginning and end of each subroutine must be within the range of the calling program, and the offset to IX or IY must be within the range of the end of the subroutine. If the offset is outside the range of these registers, a 16-bit offset can be created as described above under indexed addressing.

There is another option to create relative subroutines using the Z-80, but it is not time-effective. This involves a ROM call to 000B.

*Jerry Lindsly* of West Chester, Ohio, describes it this way:

When faced with the problem of relocatable code when using subroutines . . . you can use a very useful routine in ROM. Essentially what it does is this:

```
POP HL      ; Get return address to HL
JP (HL)     ; Return to calling program
```

So after you call 000B the PC is in HL. A relative call is a little more complicated:

```
CALL 00B8
LD OE,0B8H
ADD HL,OE
PUSH HL
LD OE,NNNN ; Offset from first byte
              past JP (HL).
              Use XOR A and SBC HL,OE
              if the cell is negative.

ADD HL,OE
JP (HL)
```

(Those interested in the various uses of this routine may write to Jerry at 8106 Quailwood Court, West Chester, Ohio 45069).

One could use the equivalent to such a ROM call in the program itself. This might be desirable. Certainly, if programmers had not learned it over the past two years, then the recent ROM changes (and expansions in the Model III) might dissuade overuse of calls to the Level II ROM.

---

# 5

---

## How the System Expands

Unlike automobiles, refrigerators, or shovels, computers are hardly ever completely self contained. Even if their manufacturers pretend they are making a complete unit, the machines themselves seem to desire bursting forth from their shells, commanding us to create a tabletop electronic octopus.

The reasons almost sound metaphysical: it needs more memory; it needs communications capabilities; it needs mass storage; it needs hardcopy . . . to achieve these goals, expansion capabilities were made available for the TRS-80 by means of an electronic hodgepodge called an Expansion Interface.

## The Radio Shack Expansion Interface

The Radio Shack interface originally consisted of eight major sections :

1. An extension of the keyboard's edge-card connector, initially reserved for a screen printer, but containing all the signals on the keyboard's connector.
2. Two groups of eight sockets, together with address decoding, to read and write two 16K banks of dynamic memory.
3. A decoded input/output latch for parallel printer control. A 'Centronics compatible' electronics scheme is used, with a separate edge-card connector dedicated to the printing task.

4. A set of decoded memory addresses and a LSI (large scale integration) chip for disk drive and input/output control. The disks are accessed through an independent edge-card connector.

5. A decoded output latch for dual cassette selection and control. One input and two output jacks are integral to this circuitry.

6. A decoded, input/output port for serial communications control, with accompanying terminal for an RS-232 circuit board. The RS-232 board is accessed via a separate edge-card connector.

7. A crystal and divider circuitry to provide disk access, real-time clock, and other interrupt functions.

8. Power supply circuitry.

If it seems like a lot of unrelated material to pack into a single box, then you're on the right track - the track that, for quite a while, meant trouble. The electronic clamor inside this box was to create a system-crashing din during the first year or so of expansion interface manufacturing.

The problems were manifested by memory crashes (return to MEMORY SIZE?, keyboard lockup, or, in a disk system, complete reboot and so on). There were several culprits, but the gangleader was the misconceived design itself. Putting all that material on a single board was inviting trouble, and the actual execution of the circuitry compounded it. The main difficulties

were memory refresh/select, noisy and dirty connectors, weak buffering lines, microphonics, and susceptibility to external interference.

The memory refresh/select lines were noisy and unreliable. The purpose of the refresh lines is to read the entire memory in intervals of two milliseconds or less. The memory-select lines provide the signals to select one of the 65,536 possible addresses when needed by the CPU.

In many computers, selecting a memory address is a fairly straightforward process. The CPU signals a memory request of some kind, and provides an address on the address bus. The selected memory responds by providing or accepting data according to the direction of the CPU. On the TRS-80, however, type 4116 dynamic memories are used. These memories do not have enough external pin connections to accept a complete memory address. Instead, hardware breaks the address into two parts, which are transferred sequentially.

Briefly, this is how it works. The address is sent out on the address bus. The CPU also provides a 'memory request' (MREQ) signal which triggers special circuitry. This special circuitry stands between the address lines and the dynamic memories. This circuitry produces a multiplex signal - MUX - which chooses the low seven bits of the address. It then sends those bits together with a 'row address strobe' - RAS - to the memories and reverses the MUX signal to choose the high seven bits of the address. Then the special circuitry sends those bits together with a 'column address strobe' - CAS - to the memories.

This sequence of operation provides all the address information needed by the dynamic memories to select an address. The three items most crucial for smooth memory action are the RAS, CAS and MUX signals, and as such, these should be clean, noise-free lines. Within the confines of the keyboard unit, this is the case. But once forced to travel through a cable (via two solder-coated edge connectors) into the expansion interface, the signals pick up some measure of noise. Once inside the expansion box, the noise is increased by the surrounding electronic din (remember that a separate crystal is onboard), and the requirement that these signals feed many other devices reduces their effectiveness. They become electronically tired, and memory access becomes erratic and susceptible to external electrical influences.

Thus, memory selection was impaired and memory refresh was not necessarily executed successfully in less than two milliseconds.

The next culprit in the expansion box was the lack of 'buffering'. In the *Technical Reference Handbook*, the author points out that among the requirements for hardware are that it :

4. Contains a separate power supply.
5. Does not contain more than 1 LS TTL load on any one output from the Computer.

Points 4 and 5 are very important, if you want to guarantee proper operation of your Computer.

Radio Shack, not taking its own advice, hangs five LS TTL loads on data line 0, and four on each additional data line. If an RS-232 board is installed, that becomes another load. The address lines feed as many as three devices. With a screen printer or other device connected to the expansion interface this load increases, respectively, at least once for each additional device attached!

The third system-crashing factor is an unusual one, avoided by good circuit board layout, but not always anticipated. This is covered by the rather unusual term *microphonics*. In other words, when the expansion box circuit board is tapped or vibrates, those vibrations are amplified and communicated throughout the entire circuitry by the power supply, data and address lines.

The keyboard unit can take a hefty bounce. If you're brave, give it a try by dropping it six or more inches. Chances are a running program won't crash. But rap sharply on the expansion box with the program running, and . . . you can guess. These strong pulses are not in the high-frequency range which the 'bypass' capacitors (the small disc capacitors scattered throughout the computer) can squelch; rather these are heavy, low-frequency surges which can be electronically interpreted as signal changes. Hence, the crash.

Finally, the electronics, being spread out in a plastic case, are subject to the electrical whims of the outside world. Seated directly below the video monitor, the expansion box is susceptible to any strong noises contributed by the monitor, or through the power lines and rebroadcast by the monitor's circuitry. Printer heads and motors, disk drive motors, and what have you all let the expansion box know they are operating. So it responds by crashing its sensitive and overworked memory circuits.

Several solutions have been created. The first was that bulge between expansion box and keyboard unit: the buffered cable. The second was the 'twisted pair' modification, another bulge between the boxes. The last was a

rethinking of the expansion circuitry and redesign of the board. The last was the only one that worked.

The idea behind the buffered cable was sound - to strengthen the overworked signal lines by having them feed a single integrated circuit each, and then have that integrated circuit feed the expansion box. Furthermore, 'termination resistors' could then be added, which work something like this: power supply lines contain bypass capacitors to squelch noise, but placing capacitors on the signal lines would slow down important signals as well. Instead, low-value resistors could make the signal lines 'hug' the power supply and ground lines, increasing their resistance to low-level, transient, electronic noises.

First installation of buffered cables were less than successful, partly because the three memory-select signals were very fast signals delayed too long by a combination of buffering and transmutation inside the expansion box. Thus, these lines were pulled outside the 40-pin cable via a three-pin DIN connector, separately terminated. This kept them up to speed, but isolated them from the accompanying noise.

Nevertheless, these modifications were only making up for difficulties inherent in the expansion box design, and could not improve the box's susceptibility to noise all around, or its microphonic tendencies.

The solution was to put the buffers on board the expansion interface itself, and redesign the board to reduce interference. New memory-select lines were created by the RAS line alone to eliminate noise in these signals, but the method used was to create problems concerning speed modifications (which, of course, Radio Shack did not authorize and could not consider in its redesign - see the way out of this problem later in this Chapter).

These difficulties, though, should be placed in perspective. When connectors are well cleaned, good memories are used to stock the expansion box, and the computer's environment kept clean, static-free and vibration-proofed, even the first, unbuffered systems work successfully. My own system (on which this book is being prepared) is a 48K, one-disk system with RS-232 board, printer, two modems, *Exatron Stringy Floppy*, two cassettes and home-brew interfaces - all connected to an *unbuffered* expansion box of early vintage. But it must be cared for. The designers were simply not prepared for the TRS-80's ultimate home and business environment.

One increase in reliability is not mentioned in the Radio Shack redesign. Electronic parts are 'derated' according to temperature. That is, as temperature increases, their performance decreases. Some circuits, like 74LS types, are strong and steady right to the limit; others, particularly dynamic memories, become flakey as the heat builds up. Thus, reliability of the expansion box can be improved by removing the two power supply transformers from the expansion box.

## Solving the Ground Problem

I call this a ground problem, but it really refers to the microphonic tendency of the expansion board. The first and easiest solution is to remove and cushion the expansion circuit board. One principle of vibration-proofing is that when different materials are sandwiched together, a certain amount of sound and vibration energy is reflected back from the junction of the different materials.

Thus, all sources of potential vibration should be cushioned. Set a thick cloth (like terrycloth) on your computer desk, and a wooden baseplate atop that. On it, place a mat made up of several thin layers of plastic pad, rubber foam, and cloth to a thickness of about 1/4 to 1/2 inch. Assure that the keyboard unit and interface seat comfortably in them (you might drill shallow depressions in the wood for stability), but that there is some air space for cooling. Remove all the plastic doors which attach to the interface and discard them. In their place put rectangles, made of alternating layers of cork and styrofoam, which snuggle firmly between the connectors and the upper and lower walls of the case. See Figure 5-1.

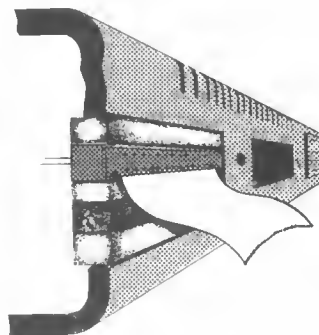


Figure 5-1. Shockproofing diagram.

Other sources of vibration are the cables that attach to the expansion box. Rather than have cables vibrate and swing with a printer, or stretch when you readjust the position of a disk drive or modem, sandwich each one under a leather (or synthetic plastic, if you prefer) bridge, and screw

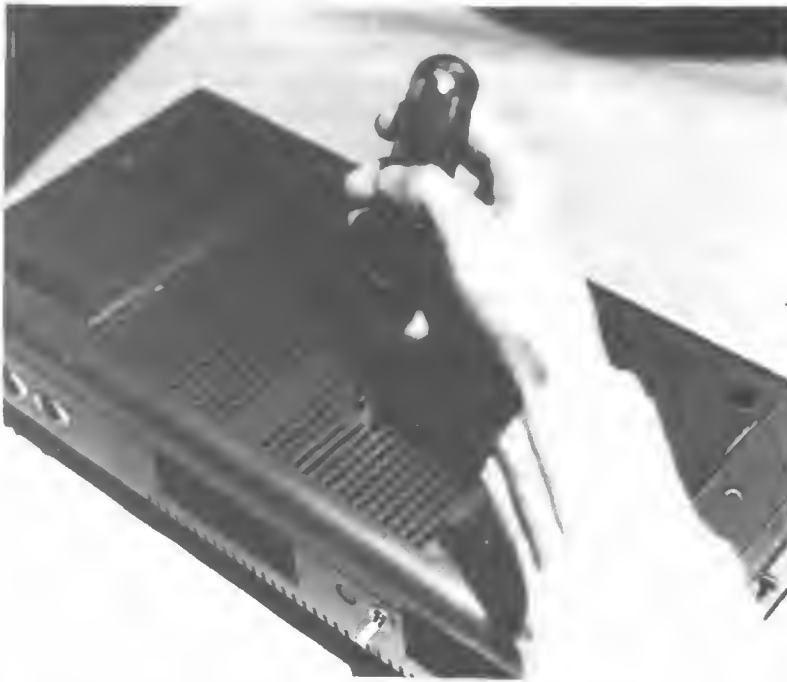


Photo 5-1. Photos on opening the expansion box. Screws are removed from bottom; note that like the CPU, a warranty-warning label may cover one screw.



Bottom is lifted off to reveal top of expansion circuit board. Screws holding circuit board are then removed.

the ends of this bridge through the cushion pad into the wooden base. When jostled, the cables should not move between the bridge and the expansion box.

Finally, avoid the possibility of stretching the keyboard/expansion box cable. The best way is to drill shallow depressions in the wood base, as noted earlier.

These are not specifically electronic hardware corrections, because only a thorough redesign of the expansion interface can truly cure these microphonics problems.

### Expanding to 32K and 48K Memory

Memory expansion in the interface itself is quite simple. Remove power from the interface, disconnect cables, and lift out the power supply transformers. The latter action is necessary because the power supplies will only fall out anyway when the bottom comes off. Flip the expansion box over, and remove the six black Phillips screws. Lift off the bottom cover.

There are sixteen unfilled sockets on the board, reserved for memory expansion (see Photo 5-2). The *higher-numbered* sockets (Z-9 to Z-16) will be used for the first block of memory, for a 32K total RAM. The lower-numbered sockets expand the system to the complete 48K.

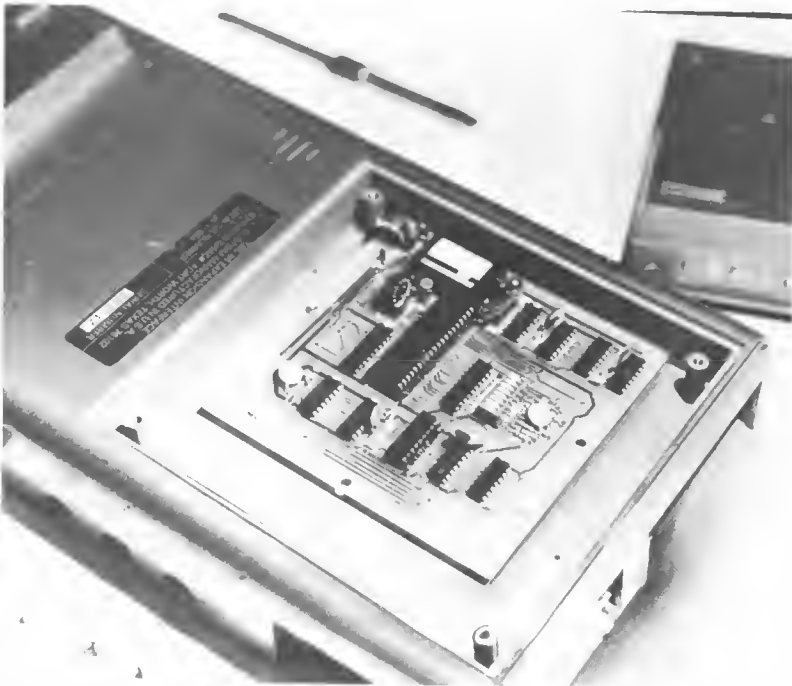
The memories to be used are type 4116, as used in the keyboard unit (see Chapter 4). There are some differences, though. Oddly, some sources claimed that slower memories seemed to work better in the older interfaces, but fast 250 nS RAMs are the most commonly available now. I have not come across an expansion box of any vintage that could not take fast RAMs like these, including my own, which dates to early 1978.

First, refer to Chapter 3's notes on handling static sensitive integrated circuits. Take each RAM and orient it with its notch or dot at the top facing in the same direction as the notch in the sockets (and in the same direction as the rest of the integrated circuits on the board). Fit each one of the first eight chips carefully in place, being sure that the pins slide into the sockets and neither buckle underneath the IC nor slip outside the socket edge. Press the memory chip gently but firmly in place.

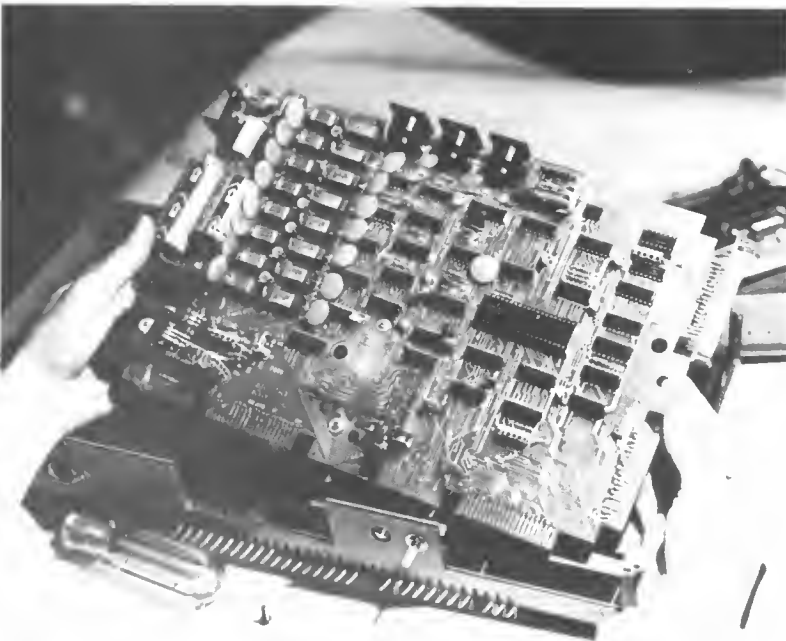
Now, before installing another 16K memory block, test this group. Flip the expansion box over (the cover need not be installed), reconnect cables and power supply, and power up the expansion box. While holding (BREAK), power up the computer itself, and press (ENTER) in response to MEMORY SIZE. Now PRINT MEM, and a figure of 31956 will be displayed. If all is well, power down, remove cables, and

insert the remaining eight memory chips. Again restore cables and power, and go through the initialization procedure. Memory should now read 48340. Replace the cover and cover screws, and you're set with an expanded system.

Problems? If there are any, they will likely fall



*If RS-232 board is in place, the two screws holding it must also be removed.*



*Entire expansion card lifts out easily.*

into these categories:

1. PRINT MEM returns 15572 or some figure much below the expected 32K value.
2. The system continuously crashes back to MEMORY SIZE? or just locks up. This happens most often while running any program.
3. The system just locks up with garbage on the screen at power-up.

Well, the last first. Don't forget to hold the (BREAK) key down at power-up in a non-disk system. The other two are tougher.

If PRINT MEM returns 15572, you might have installed the memory incorrectly. Keep booting (press the Reset button while holding down (BREAK)), and hitting (ENTER) in response to MEMORY SIZE? PRINT MEM, checking over and over. If a figure higher than 15572 is ever returned, you might have expansion box problems.

Try entering a number between 15580 and 32700 in response to MEMORY SIZE?. If it is ever accepted, chances are you've got the memory chips installed correctly. Try POKEing and PEEKing, like this :

```
POKE 25000,100 : PRINT PEEK(25000)
POKE 26000,103 : PRINT PEEK(26000)
```

and so on. If the value you POKed is the same as what you PEEK at, the memory is there, but it's not acting reliably. Here are a few solutions (see also the expansion box general solutions above) :

1. Clean cables, connectors, etc., and shock-proof the unit.
2. Remove the buffered cable if you have one, and replace it with a regular one. *Before you do this*, open the expansion box and find pin #1 on the edge connector. Cut that trace, or remove the wire you see there. This is the 5-volt power supply to run the buffering box. The matching pin on the keyboard unit is a ground!
3. If you have a newer expansion box, make the high speed alteration noted above, although you shouldn't be having these problems with the new box.
4. You actually might have some flawed memory chips, though this is unlikely. Switch those in your keyboard unit with those in the expansion box, and see if the problem goes away (or the keyboard unit doesn't work now). Run the memory test (Chapter 3) to find out which chip(s) may be bad.

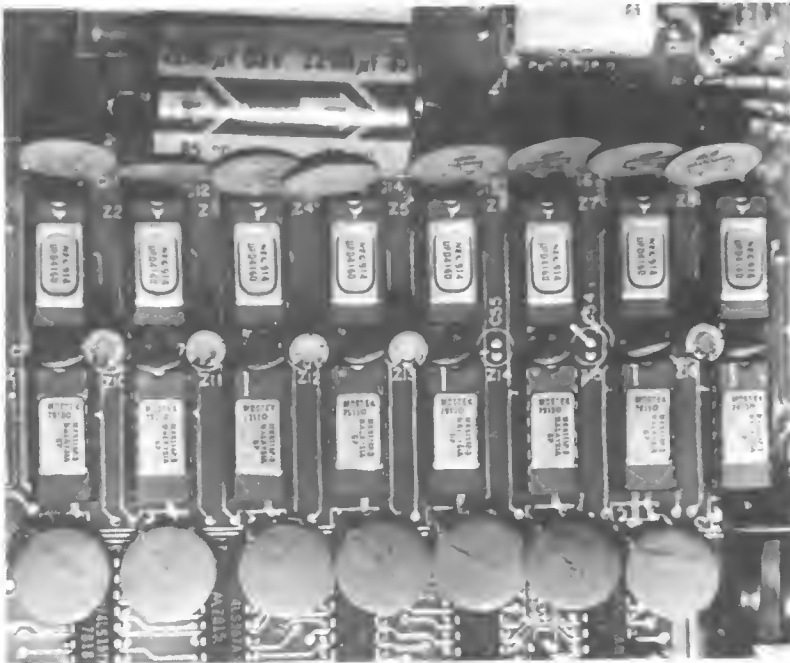


Photo 5-2. Expansion interface memory sockets. 32K memory fits into sockets Z1 through Z16. Oddly, Tandy engineers did not install power supply filter capacitors C54 and C55, nor are they included in the schematic drawing.

5. Make the small change to Z69 in the keyboard unit, as described in the 200% speed-up modification.

The options are few when dealing with an older expansion box. They will work, but you have to provide them with ideal conditions. Which points out the solution to the possible program crashes noted in problem number 3 above. Pamper the expansion interface when a program is running. Don't have the kids playing kick-the-cat near the computer, or the dog jumping about and thumping a large tail next to you, waiting to be fed. Keep the washers, driers, and mixers in the vicinity switched off.

### Speeding Up Newer Expansion Boxes

Radio Shack's cures for the memory refresh/select difficulties in the expansion interface created a new problem for those who wish to increase the speed of the computer. All three select lines (RAS, CAS, and MUX) are not used in the revised expansion box; instead only RAS is used.

Technically, the RAS line is sent thrice through an inductance network which provides three differently delayed doppelgangers of the original signal. These are tweaked back into digital shape by a buffer circuit, and the three are

labeled MRAS, MMUX, and MCAS. The ploy works.

Trouble is, these lines don't care about the speed of the central processing unit. Fast or slow, the false MMUX and MCAS signals trip along at a fixed rate after MRAS. The only cure is to somehow speed up that rate and buy fast, high-quality memory. The speedup is accomplished by *not* delaying the RAS to produce MRAS, and creating MMUX and MCAS in the old MRAS and MMUX positions, respectively. The faster memory (200 nS) is up to you.

To provide the speedup, you'll need to cut some traces and run some wires. Remember, this adjustment is for newer expansion boxes only – the ones that the sales clerk says “don't need the buffered cable”. Here is the order :

1. Find Z37 and Z38.
2. Cut the trace leading from Z38, pin 9. Call the end furthest from Z38 ‘Trace A’.
3. Cut the trace leading from Z38, pin 8. Call the end furthest from Z38 ‘Trace B’.
4. Cut the trace leading from Z37, pin 4. Call the end furthest from Z37 ‘Trace C’.
5. Attach Trace A to Z38, pin 11.
6. Attach Trace B to Z37, pin 4.
7. Attach Trace C to Z38, pin 9.

The modification is complete. Close the box and power up again; the system should work normally (if not better as time goes by). Both higher speed modifications presented in Chapter 4 will work with these changes, so long as your memory is the faster, 200 nS type. If the memory seems at all unreliable, double-check your work. If the work is fine, then your memory may not be up to the change.

### LNW, Microtek and Other Expansion Interfaces

The first of the TRS-80 products to be attacked, and rightly so, was the flakey expansion interface. By the time Radio Shack was offering its initial hopeless ‘fixes’, a trio of Californians were redesigning the box for their colleagues in a TRS-80 user group. This was to become the premier challenger to the expansion interface, the LNW System Expansion, priced at a mere \$70 for the bare board, circuit diagrams, parts placement, and a few additions to the Tandy system.

The LNW System Expansion is essentially the Radio Shack expansion interface with a new

## Using Those 4K Leftovers

Did you insist on keeping the 4K RAMs which were removed from the keyboard unit when your 16K RAMs were installed by Radio Shack? Or did you do the installation yourself? If so, you've got 4K of dynamic memory worth about \$20 off the shelf. But the trouble is you can't use them in your expansion box, because it's set up for 16K memories only.

But, if you want to spend a bit of time and can't afford an expansion interface, you can extend that PRINT MEM to 19668! Almost

everything is in place already, so all you need is an address decoder and multiplexer circuit.

The circuit presented below can be wire wrapped on a small board, and will provide that extra 4096 bytes. The cost? About \$5 for circuits and sockets, another \$5 for an edge connector (use the *Texas Instruments* wire wrap type sold by *Digi-Key*, part number C6-20), and some time.

When you begin to fill your expansion interface, though, you'll have to give up the 4K RAM extension, because the first block of 16K memory in the box takes those memory addresses.

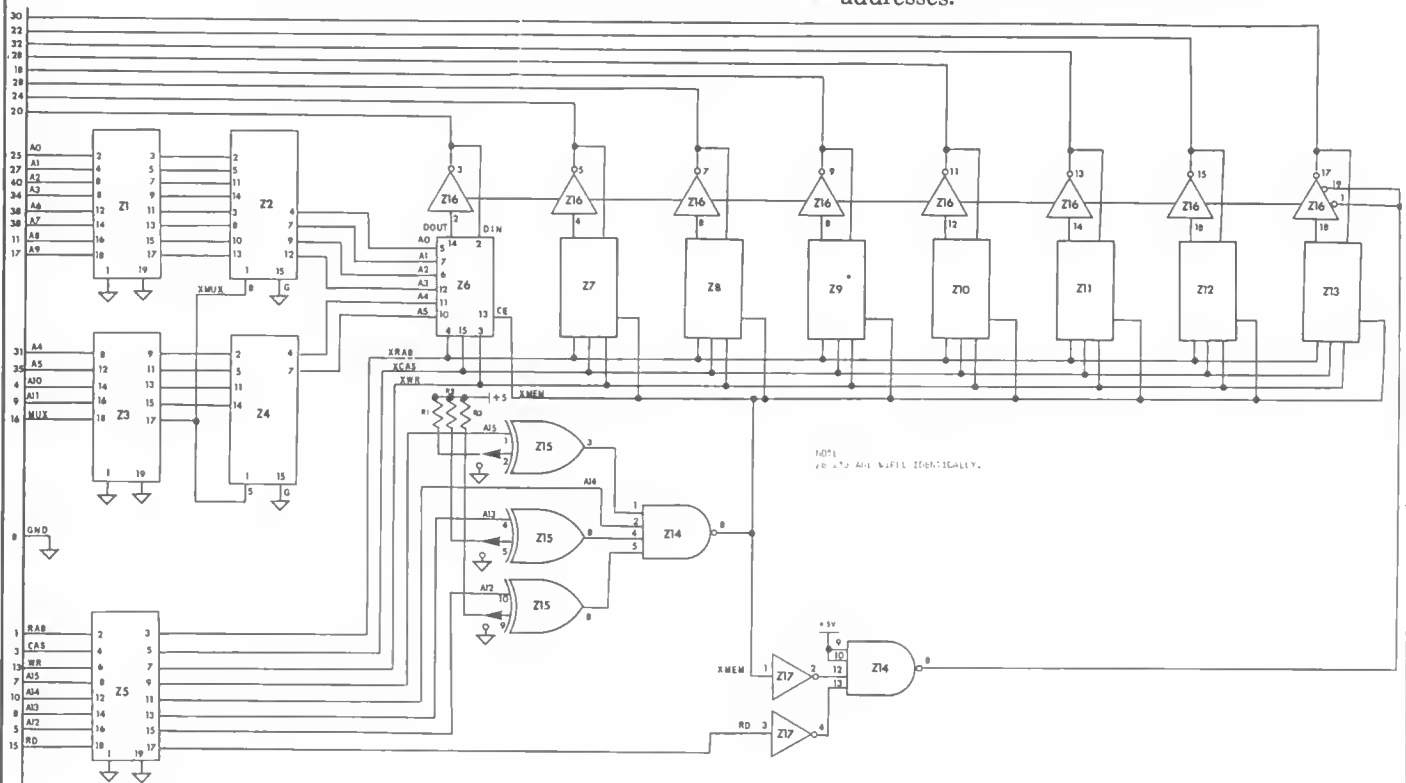
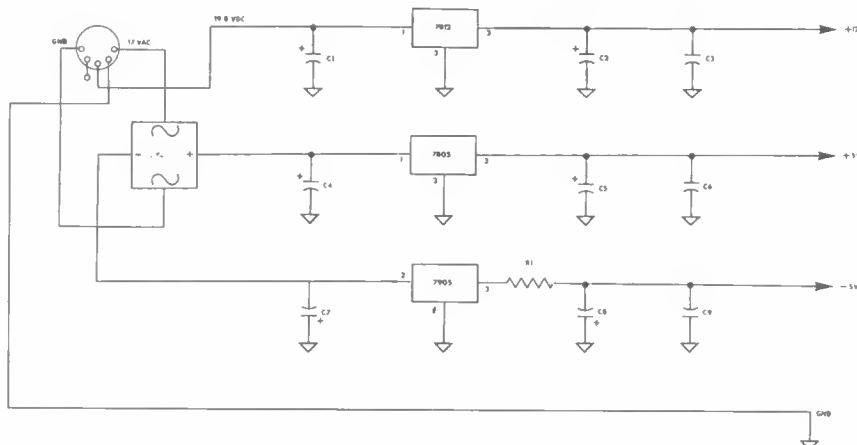
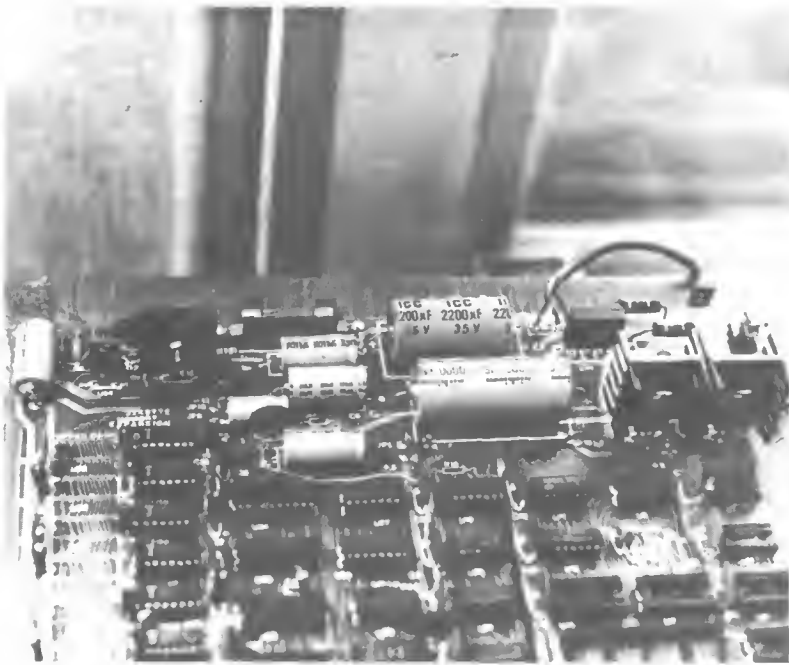


Figure 5-3. 4K dynamic RAM add-on circuit.





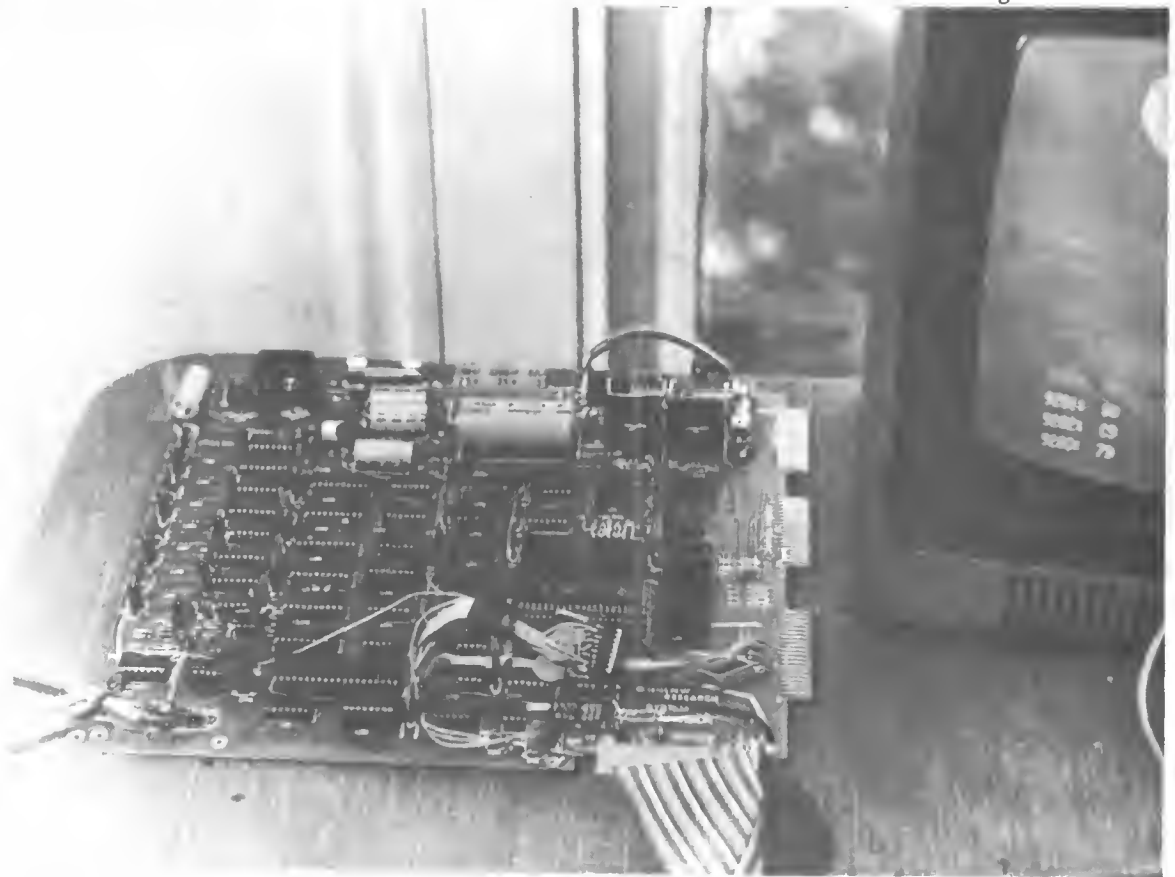
*Hefty power supply includes fuse and multiple heat sinks, unlike TRS model with inaccessible fuse and light heat sinks.*

layout, on-board buffers (which appeared finally on Radio Shack's own product in the spring of 1980), standard RS-232, and noise-reducing passive bus termination. This last was of particular importance in making the LNW board noise-free and consequently crash-free.

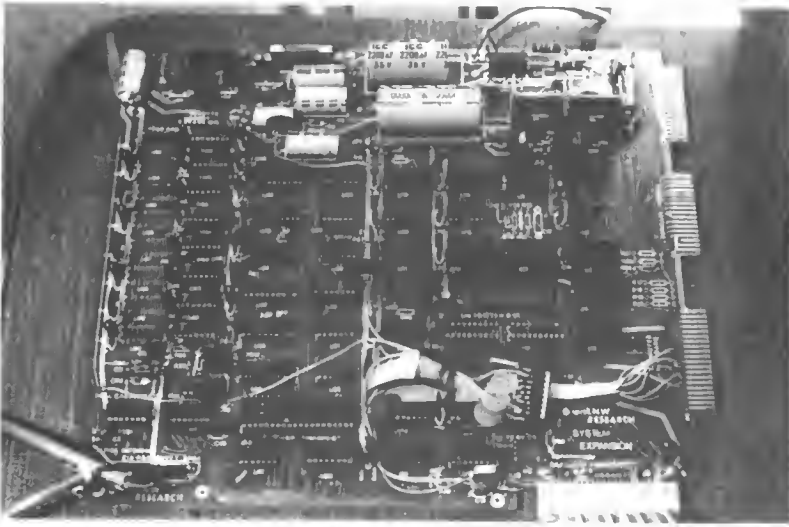
As with the Radio Shack expansion, the LNW board has sockets for 32K additional RAM, but also suggests that sockets be used for all parts – good advice, especially since chips such as the open-collector buffers used in the disk line driver need occasional replacement.

This board is also modular in a way that the complete, boxed interface from Tandy could not be. Only as much as the user needs at one time may be installed. Thus, the \$25 disk controller chip can be left out of the system until (or if) a disk drive is added; so also the disk's peripheral chips can be omitted. Where interrupts are not desired, the entire crystal controlled clock can be excluded. Less than \$150 gets the user a bare-bones 32K memory expansion.

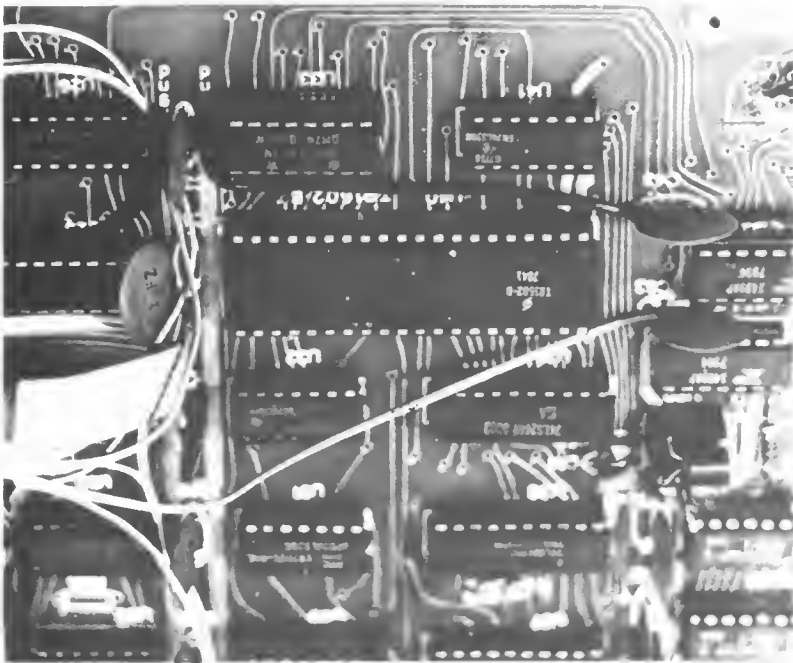
The trick to the LNW board is dexterity. This is not a project for total novices, because it involves extensive soldering and some



*LNW Expansion compared with TRS-80 monitor. Board is approximately the same size as standard TRS expansion interface.*



*Communications are built into LNW expansion; TR1602 UART provides serial capabilities. Note clear marking of all parts, including an industry part number at the UART socket.*



*Virtually complete LNW board in use. User has added programmable baud rate generator (bottom right) in place of hard-wired jumpers recommended by LNW Research.*

knowledge about what the parts look like. There is no Heathkit-style check-off list. You get a parts layout, schematics, and encouragement. But you also get a noise free, reliable expansion interface.

For those who plan to build (or have already built) the LNW board, a few changes might be in order. If you wish your reset button to work, you must not only make a change similar to that for the Radio Shack expansion box, but you must also change the values of the pull-up and pull-down resistors. Change the pull-up resistors on the data lines to 470 ohms, and the pull down resistors to 680 ohms. This will provide the desired 11111111 signal when the reset key is hit and the disk buffer-disable modification has been made.

To make the mod, turn to the expansion interface reset recovery presented in Chapter (?), and follow those same directions, but using the integrated circuit on the LNW board marked U19.

## Special Section: Two Other 80s

### The PMC-80.

As production of the TRS-80 Model I was winding down at Tandy, the rumored 'Hong Kong Copy' made its appearance in the United States, under the name PMC-80, and sold by Personal Microcomputers, Inc., of Mountain View, California.

I was personally ready to greet the PMC-80 with much skepticism, expecting a weaker TRS-80 with little to commend it and a lot to avoid. But, tacky advertising aside, the PMC-80 is a functional, satisfactorily designed product. In fact, because of the experience of the original '80 itself, this copy is probably better designed than Tandy's product. That's probably an indirect compliment to both Fort Worth's foresight and Microsoft's Level II BASIC, which the PMC-80 contains virtually byte for byte.

First, the obvious similarities :

1. Z-80 microprocessor running at 1.77 MHz.
2. 64 x 16 screen resolution, with 128 x 48 graphics.
3. Video monitor output compatible with the Radio Shack monitor.
4. Microsoft Level II BASIC in a 3-chip ROM set and standard type 4116 16K memory.



Photo 5-3. The Z-80 CPU — Used in the PMC-80.

And then, the differences :

1. Built-in cassette player with no level control necessary, with provision for two via a second cassette jack already in place.
2. Video RAMs are in sockets, all seven are installed for lower case use, and an industry-standard character generator is used. Lower case is not provided, but is available.
3. An oversized, well-ventilated power supply feeds the system.
4. Channel 3 RF video is provided, along with a connector cable and adaptor.
5. Keyboard has a cassette motor on/off control, but is missing clear and right-arrow keys. These can be installed by the user on some machines.

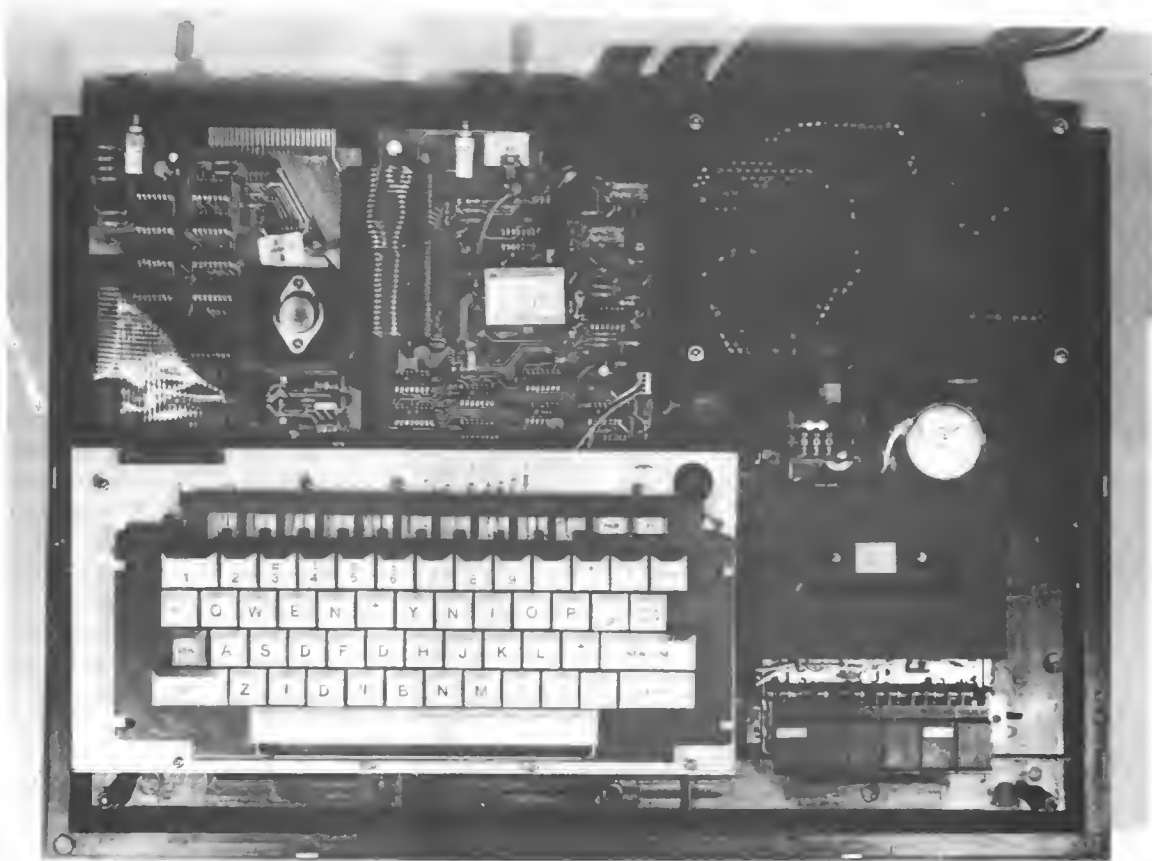


Complete PMC-80 system can include CPU unit with cassette, printer, expansion box, monitor, and disk drives. Photo courtesy of Personal Micro Computers, Inc.

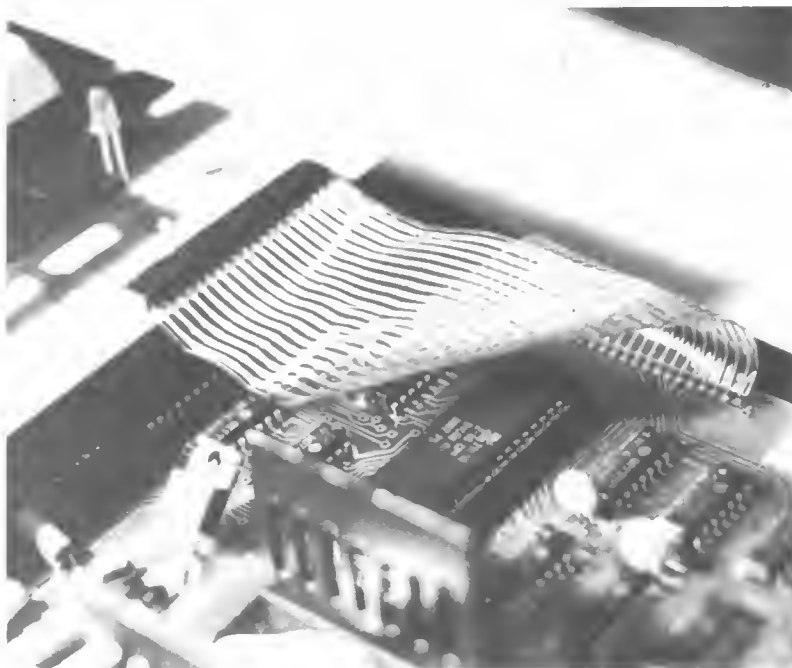
6. An odd 32-character mode, selected differently (with a switch). Instead of alternate letters in video memory, the 'left half' of video memory is used. Normal CHR\$(32) is shown instead as normal-size letters appearing alternately on the screen.
7. The keyboard types easily, with no evidence of keybounce.
8. The expansion connector is gold-plated, but the inter-board and keyboard connectors are just solid wires pushed into receptacles. Heavy duty relays are used for the dual-cassette I/O.
9. The gold-plated expansion connector, alas, is not a TRS-80 compatible 40-pin type. The PMC-to-TRS adaptor is an option.

Are the TRS-80 and PMC-80 software-compatible? From what I can tell after running both BASIC programs and machine language utilities employing calls to ROM: yes, with minor reservations. First of all, the missing clear and right arrow keys are a downright pain; users of programs such as Scripsit or the Penmod version of Electric Pencil are back in the good old days of having to add keys to the system to get these programs to work.

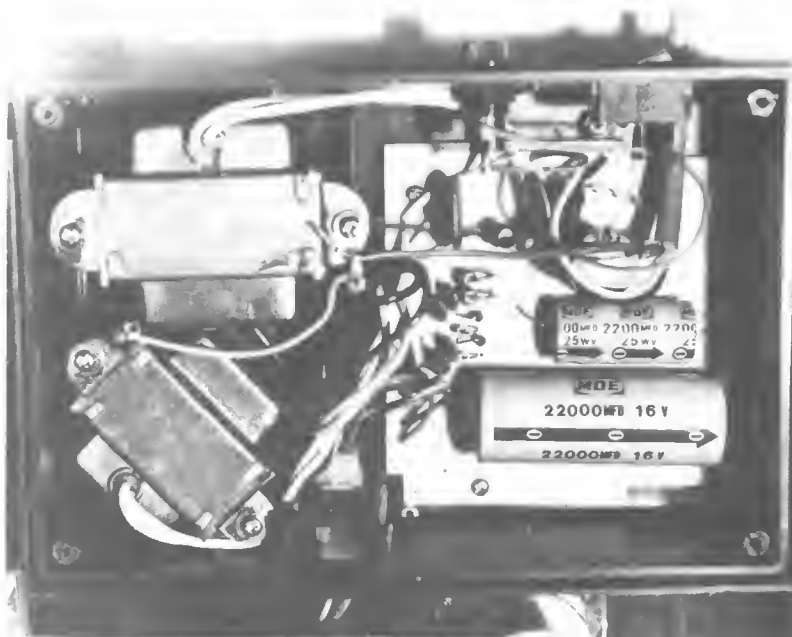
Major ROM calls, though, seem to be intact, so there's little worry about compatibility there. The only programs that seem to get lost are those that call inside routines, or that call partial routines in ROM. BASIC runs flawlessly, as do mixed BASIC/machine programs.



*Inside the PMC-80. Keyboard, two electronics boards, cassette player, and power supply are attached to the baseplate. Multiconductor wire jumpers plug in place.*



Keyboard connector attaches to main board via a removable cable, a distinct improvement from the permanent TRS-80 cable. Note Z-80 CPU underneath cable.



Hefty power supply provides drive for computer electronics as well as a separate supply for the cassette recorder. Fuse is user-replaceable, in a standard socket.

## Other comments :

The video display is cleaner, and the letters are better formed, prettier, and with no touch of blurring, even on the inexpensive TRS-80 monitor. They are quite similar to the better quality characters of the TRS-80 Model III. The lowercase option was not installed in the unit I used, but since it uses the standard character generators and has a clean video output circuit, similar well formed letters can be expected.

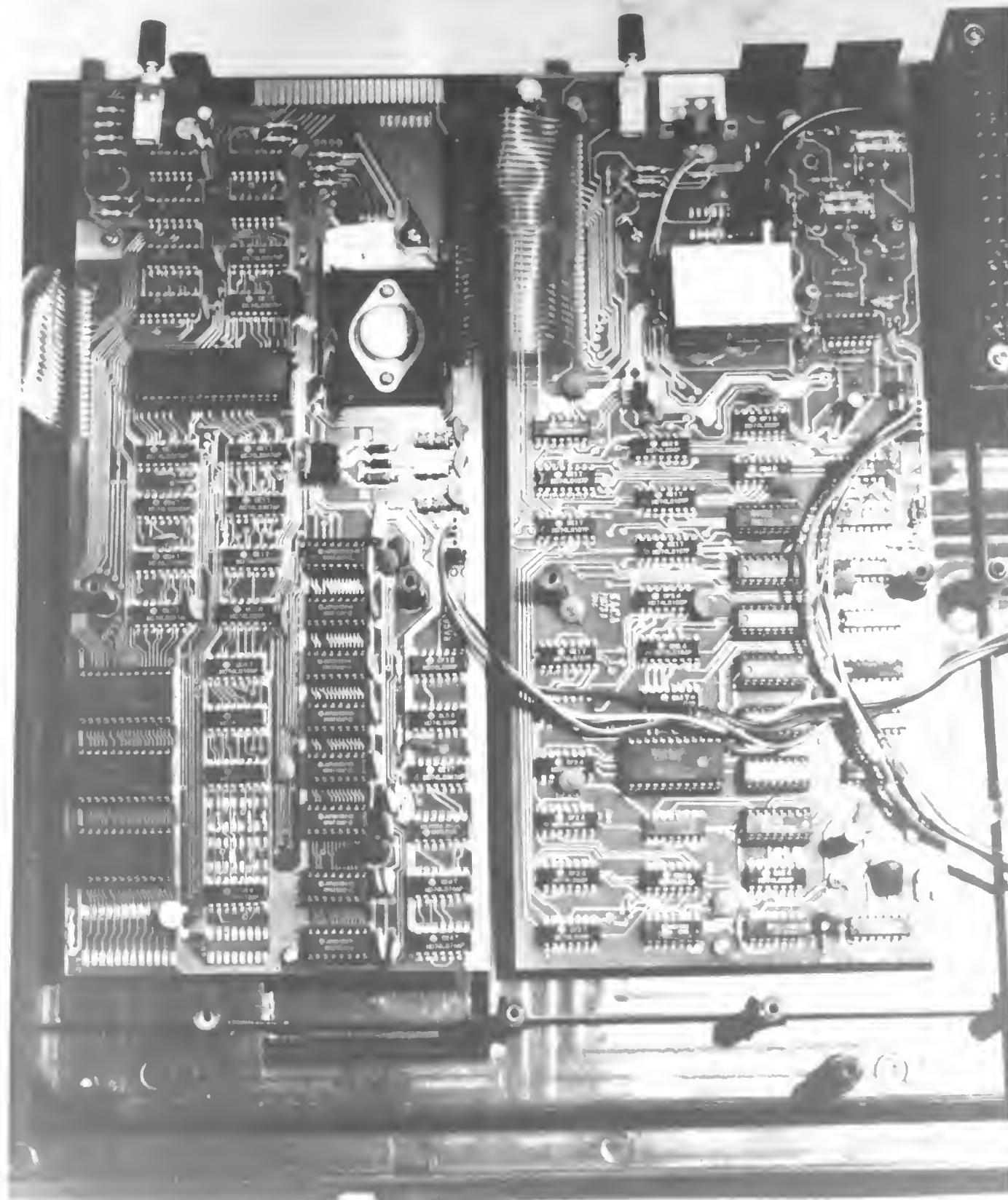
Cassette input/output is sweet. Problem programs loaded easily with this system. The playback of the cassette deck is digitized with LM324 comparators instead of 3900 Norton amps, and so the digitization is more successful.

The oversized power supply shows every sign of providing plenty of surplus current. There is no need for power supply adjustment in this system, because in place of the adjustable 723 regulator used in the TRS-80, fixed 7812 and 7805 regulators are employed.

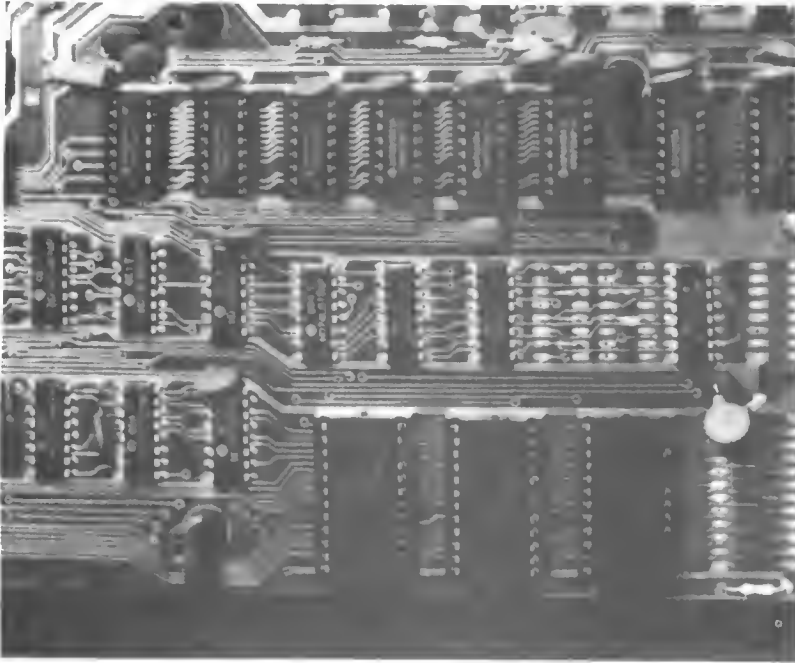
The unit, though heavier than the TRS-80, is more complete in the sense that it has a built-in cassette player and RF video output. For traveling, the PMC-80 makes much more sense, because a video monitor can be used in long sessions at home or office, and an ordinary television can be used at other times. The case is tacky, especially the fake walnut grain plasticoid sides, but then the down-to-business futura TRS-80 was no great aesthetic shakes, either. The keyboard is placed at a comfortable angle, and the added weight and size gives the unit a more responsive typing feel.

## The LNW-80 Kit.

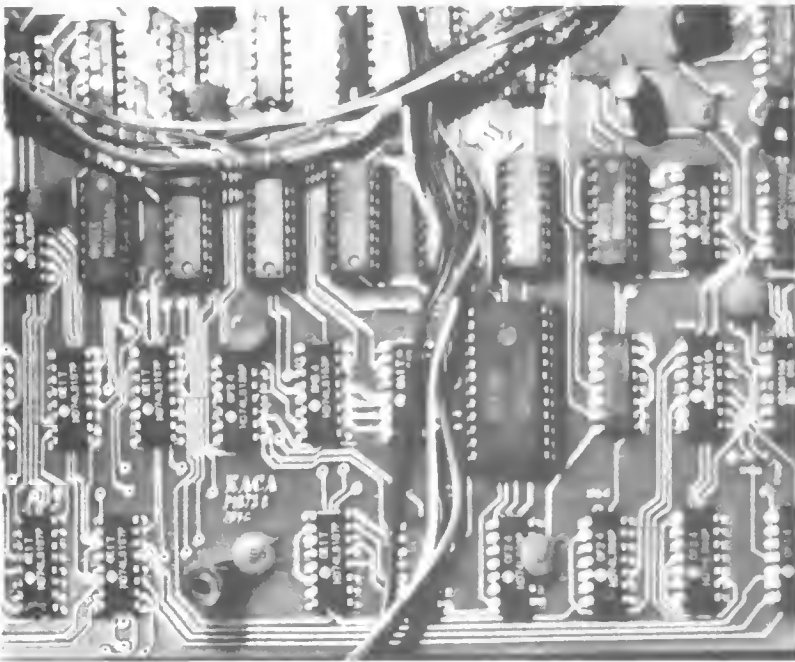
This is a welcome newcomer, providing the advantages of the Model I with LNW's well respected circuit board. As with their system expansion, this LNW board is sold naked. The user contributes the parts as required.



*Under the keyboard is the complete electronics control.*



*CPU control card contains 3-chip Level II ROMs, 16K dynamic memory, and power supply. Circuit board design isn't pretty, but is very reliable.*



*Video display card includes 7-bit video memory (unlike original TRS-80 6-bit memory), industry-standard CGR-001 character generator, RF modulator, and sockets for video memory chips.*

## CPU Interruptus

The interrupt is a programmer's mystery tool. On large machines the interrupt controls are so sacred that they are available only to the system engineers. Without a password, assembler commands directed toward interrupts are generally ignored.

TRS-80 users might have in mind some dreams of the Z-80's interrupt power. They've heard of it. Allusions have been made to it. Yet because it operates outside familiar programming bounds, it is even now a mystery.

Furthermore, this may be one of those cases where the Tandy engineers seriously erred in the concept, (rather than in the design, which problems we know all too well). The inclusion of additional interrupt power could have been done for pennies.

## Whatsis?

The activities of a computer program can be viewed in relative rather than real time. In other words, a computer performs its tasks in the order that the programmer has chosen, with a timing that is inherent to the oscillations of the computer's master clock. Change the spacing of those oscillations, and the absolute time – the time with respect to 'real' time – changes, yet the relative time within the computer program is still the same.

Unfortunately for timekeeping purposes, changing either the clock of the computer, or changing the way in which the computer handles its tasks, may actually cause a shift in the real time reported to the user. An interrupt, then, is electronic shoulder tapping, the ability of a device external to the microprocessor and its respective memory to demand the CPU's attention for imperative tasks. For example, an external pulsing clock says to the CPU, 'Okay, I don't care what you're doing, stop it now and take care of updating of the real-time clock.'

Such is the simplest use of an interrupt. And, for a change, the computer jargon term is true to its actual purpose. It interrupts the operation of the central processing unit, demanding attention.

An interrupt is akin to a ringing telephone. As long as you are within earshot, the ringing of the telephone interrupts your activity. Whether you choose to respond consciously to it is irrelevant to the actual interruption. If you can hear, and if you are within hearing range of the bell, it does

indeed influence the activities of your nervous system.

How you respond to that ringing telephone – whether you ignore it, whether you go to it and answer it, or whether you merely swear at it – is what might be called, in computer terms, ‘servicing the interrupt’.

The Z-80 microprocessor contains a wide range of ways in which an interrupt may be acknowledged, received, or serviced. You respond to a knock, doorbell, telephone, child’s cry, or gunshot with various levels of urgency. If more than one were to occur about the same time, you would be forced to mask out the one of lower priority and give your attention to the more pressing interruption.

Before describing the ways the Z-80 microprocessor might respond to an interrupt, it should be noted that an interrupt is uniquely different from the computer’s usual input/output schemes, just as the ringing telephone or gunshot are considerably different from checking the color of a traffic signal as you approach it; inspecting the mailbox as you arrive home; or scratching the cat.

In these analogies, your ears most often serve as your human interrupt line. As long as they are ‘in the circuit’, they will interrupt you. You can close your eyes to the traffic signal, ignore the presence of the mailbox by averting your glance, or cold-shoulder the cat (at your own risk), but (lacking earflaps), sound will give you an appropriate kick in the eardrums.

Back to the Z-80’s interrupt schemes. During a movie, were someone to yell “Fire!”, chances are that everyone – barring suicides and existentialists – would respond to that interruption with their personal interrupt service routine. Most likely that routine would be to leave the theater quickly.

An equivalent reaction by the Z-80 is produced by a signal applied to its ‘non-maskable interrupt’ (NMI). This interrupt is always acknowledged. The CPU cannot ignore it. On the TRS-80, it is this interrupt which is activated when the Reset button is pressed.

The NMI is hard-wired into the system in such a way that we have no control over it, so that if these sentences were a program ...

```
READY
>_
```

... and not only that: the execution of a HALT instruction produces a ‘Halt Acknowledge’ output signal which the Tandy Engineers have

OR-gated with the NMI to ...

```
READY
>_
```

... yup, you guessed it.

Contrarily, the Z-80 has a maskable interrupt (INT). It is an interrupt line which, when activated, may either be ignored or acknowledged by the processor if the programmer has so specified.

Because their activities are not fixed, these maskable interrupts are the ones of interest to us. Although, as I’ve said, the Z-80 was designed with a wide range of interrupts, that wide range is not accessible to TRS-80 users because the Tandy engineers did not design the TRS-80 in a way that could make those interrupts available.

Nevertheless, here is a look at the Z-80 interrupt modes.

**Mode 0.** This mode allows the interrupting device to place a machine language instruction directly on the data bus. The CPU will then execute that command rather than the next instruction in the normal program flow. Designers of the Z-80 suggest that a restart (RST) instruction be used.

Restarts have been covered in *The Alternate Source* and other publications, but briefly, they are single byte subroutine calls especially designed for this kind of interrupt operation. Normally, subroutines require specification of a CALL instruction (CD) and two absolute address bytes. The RST instruction is a single byte command causing the CPU to stash the program counter on the stack and move to one of eight locations in low memory.

**Mode 1.** Upon interrupt, this mode forces the CPU to preserve the program counter and then move directly to location 38.

(line to absorb indent)

**Mode 2.** This is considered the most powerful interrupt response mode of the Z-80 because “the programmer maintains a table of 16 bit starting addresses for every interrupt service routine ... When an interrupt is accepted, a 16 bit pointer (is) formed to obtain the desired interrupt service routine address from the table. The upper 8 bits of this pointer (are) formed from the contents of the I register. The I register must have been previously loaded with the desired value by the programmer. The lower eight bits of the pointer must be supplied by the interrupting device....” (from Z-80 Technical Manual).

Before you permit excitement (or even a raised eyebrow) about the last interrupt mode, it should be pointed out that neither Mode 0 nor Mode 2 may be used on the TRS-80, for this hardware reason: in order to place any instruction or address on the data bus, it must be possible to configure the data bus as an input. Within the internal circuitry of the TRS-80 itself, the data bus is fully bidirectional. This is normal, because the data bus is in an *input-to-CPU* state whenever memory is being read or input instructions are being executed, and in an *output-from-CPU* state whenever information is being stored in memory or output instructions are taking place.

Unfortunately, only the data output is brought to the edge card of the TRS-80. There is no buffer wired to the edge card which is pointed *inward* to the CPU, activated when the IORQ signal is sent from the CPU during interrupt! Therefore it is impossible to place either an instruction or an address byte on the data bus from outside the computer.

(Of all the possible under-utilization of the Z-80's power in the TRS-80, this is to me the most discouraging. You can cure something like the stingy lack of an extra video bit for lower case, but adding a second set of buffers requires a true hardware hacker. You might as well build a new computer. *Aveatquevale*, Model I.)

Thus, the balance of this section will be concerned exclusively with interrupt Mode 1 (set by executing the command IM1). As noted, the execution of this instruction forces the CPU to address 38.

Address 38 has also been usurped by the TRS-80 designers by placing it in ROM (read only memory). Because the first act of the instruction at 38 (which incidentally is the last of the RST instructions) is to jump into RAM at 4012, it can nevertheless be accessed easily.

With a Level II CPU unit alone, the edge-card connector is brought out so the interrupt line is present. However, if you are an expansion interface owner or disc user, then you should know that the interrupt itself is dominated by the expansion interface unit's clock, which places a constant 25-millisecond pulse on the interrupt line. Therefore, not only are interrupt modes 0 and 2 made unavailable by the TRS-80 design, but (unless you are willing to cut a trace) the presence of the expansion interface limits even the use of Mode 1.

Despite all these limitations (whew!), the 25 mS interrupt does allow options beyond just the

real-time clock.

Which brings us to the reason the Z-80 has so many interrupt modes. The most powerful kind of interrupt places an instruction or address on the bus, directly influencing CPU activities every time it occurs, but only when it is needed. If the hardware is well designed, this means every interruption is essential, and is handled expeditiously.

On the other hand, 'polling' interrupts for service requests is wasteful. That is, when there is a single interrupt mode which all devices must use, then the CPU is forced to ask each one, 'Did you interrupt me? No? Well, did you interrupt me? No? Well then, did *you* interrupt me? No? Okay, then . . .', until it finds the culprit. It not only services it, but also checks the remainder of the devices in case other interrupts occurred simultaneously with the first, or during the interrupt service routine.

We are therefore limited not only to a single interrupt mode, but also to a required polling technique, because this 40-times-per-second pulse is coming through without cease. No real 'device' is asking for service, but the effete request is present nonetheless. Yet we can still use it. Here's how: when the interrupt comes through, the CPU sends a signal to the relatively few devices in our I/O scheme, and asks them, 'do you have any information?' If there's no information, the CPU will quickly beat a path back to normal program flow. If there is information, the CPU will service the interrupt.

## Whysis?

You may wonder what type of situation — within the context of the humble TRS-80 itself — might require this interrupt, other than the familiar real-time clock. Let's take an example of flawed human interaction in TRS-80 programming: the *Electric Pencil*.

As any reasonably good typist knows, this 'text editing' program is infuriatingly slow at the end of lines, particularly at the bottom of the page. Not only must it move the word in progress to the next line, but also scroll all 1,024 characters up the screen. If you are nimble-fingered, you can lose characters during this process.

The interrupt system might be used in order to build a buffer of keys pressed. A good typist's fastest keystrokes ('the', 'is', etc.) can be less than 50 mS apart, or 20 characters per second. As the interrupt comes through from the expansion box 40 times each second, location 387F can be

examined by the CPU. If any key other than 'shift' is pressed, its presence can be sensed by looking at this location. If its presence is sensed, the key-scan routine can be initiated, and the key information and any successive keystrokes are stored until the line break and/or scrolling is completed.

In this way the keyboard may be 'scanned' quickly, not only during entry of text, but also during I/O, such as to tape. A keystroke buffer may be built during this process (since the keyscan need only be initiated if a key is pressed) between output bytes. The same is true of printing; the text can continue to be output to the printer while this activity is going on.

In fact, if a key were pressed during an I/O routine, it could be accepted into a buffer, and subsequently displayed when the next interrupt occurs. In effect, you would be time-sharing your TRS-80.

## Light Bulbs

Aha! Time-sharing your TRS-80! (Light bulbs flicker on in millions of brain rooms). Although a 25-mS pulse does not allow totally transparent time-sharing, it allows enough of it for your TRS-80 to be able to service several user requests at a time, in different ways.

But first, here's a short routine to prove that the interrupt is really there. You can assemble this, or just POKE it into memory.

```

7F00 F3      OI          ;Interrupts off
7F01 3E C3   LD          ;Get JP command
7F03 32 12 40 LD        (4012H),A ;Replace RET cmd
7F06 21 14 7F LD        HL,7F14H ;Start service
7F09 22 13 40 LD        (4013H),HL ;Place after JP
7F0C 21 19 1A LD        HL,1A19H ;Command level
7F0F E5      PUSH       HL        ;Ready to return
7F10 ED 56   IM         1         ;Set to Mode 1
7F12 FB      EI          ;Interrupts on
7F13 C9      RET         ;Return to 1A19

```

The segment above is merely a setup routine, executed once. It disables interrupts during setup, places C3 (a jump instruction) in place of C9 (return) currently found at 4012, the interrupt patch point in RAM. It then places the destination address (7F14) as the jump's operand. It pushes the return to BASIC or DOS onto the stack, sets interrupt Mode 1, enables interrupts, and returns (effectively jumping to 1A19).

Note that after an EI (Enable Interrupts) instruction, acknowledgment of any interrupts is actually delayed until *after* the completion of the instruction following the interrupt enable. Here's the interrupt service routine itself:

```

7F14 F3      OI          ;Interrupts off
7F15 F5      PUSH       AF        ;All of these
7F16 E5      PUSH       HL        ; registers
7F17 05      PUSH       DE        ; will be used
7F18 C5      PUSH       BC        ; to do this
7F19 3A EC 37 LD        A,(37ECH) ;Read disc chip
7F1C 3A ED 37 LD        A,(37EDH) ;Reset clock F/F
7F1F 21 11 01 LD        HL,0111H ;Measege start
7F22 11 25 3C LD        DE,3C25H ;Display space
7F25 01 1A 00 LD        BC,001AH ;Measege length
7F28 ED BD   LDIR         ;Display it
7F2A C1      PDP         BC        ;Put all these
7F2B 01      PDP         DE        ; registers
7F2C E1      PDP         HL        ; back in
7F2D F1      PDP         AF        ; place
7F2E FB      EI          ;Interrupts on
7F2F C9      RET         ;Return whence

```

The instructions loading the accumulator from 37EC and 37E0 accommodate the quirks of the TRS-80 hardware — the Z-80's interrupt acknowledge signal is not used in this hardware scheme. Instead, reading 37EC (disk controller status) followed by reading 37E0 (real-time clock) resets a flip-flop used to signal input from disk controller and real-time clock, respectively. This must be done each time or the interrupt will continue to be 'on', forcing the routine ever back upon itself.

You may protect memory at 32512 if you wish, as this will prevent any accidental crashes while you're trying this demo program. Also note that if you have installed a disk-disable switch on your interface, this program may initially lock up. The lockup may be cured by momentarily flipping the disable switch on then off.

So what's the point of this? Only to show that no matter what BASIC (or machine language) program you are operating, including DOS, the sign-on message will remain in the upper-right corner of the screen. CLS, scrolling, even attempting to overwrite the area with text will have no apparent effect, because the interrupt takes priority. To disable it, put a RETURN instruction in place -

POKE 16402,201

and to enable it, replace the return with a JUMP:

POKE 16402,195

Under DOS, the usual CMD"R" and CMD"T" instructions will have their intended effect. Re-initializing Level II BASIC or DOS from 0000 will wipe out the interrupt patch to 7F14.

## On to Something Useful

There are a few rules of thumb for using interrupts. First, the interrogation and service routines must be as short as possible, or else the main program will get little work done between interrupts. This means, alas, very few calls to

ROM; the ROM calls, although effective, often contain a level of detail and error-checking that may be beyond the absolute minimum needs of the service routine.

Second, every register to be used by the service routine must be preserved, usually on the stack, since there is no way of knowing which registers will be in action when an interrupt occurs. The only safe bet for ordinary BASIC programs using Level II alone is to alternate register pairs (by executing EX AF,AF' and EXX), since the alternates are not used in Level II. IX, however, must still be saved.

With these thoughts in mind, have a look at the listing below, which prints a more useful message onto the screen: the BASIC memory available at all times.

Listing 5-3. Free memory constant display.

```

7F00 F3      SETUP  OI      ;
7F01A3E C3    LD      A,DC3H ;
7F03 32 12 40 LD      [4012H],A ; Setup
7F06 21 14 7F LD      HL,SERVE ; routine
7F09 22 13 40 LD      [4013H],HL ; similar
7F0C 21 19 1A LD      HL,1A19H ; to that
7F0F E5      PUSH   HL      ; presented
7F10 E0 56    IM      1      ; above
7F12 FB      EI          ;
7F13 C9      RET          ;
3C00          VIDEO EQU 3C00H
40E8          STKBOT EQU 40E8H
40FB          ARRTOP EQU 40FBH
7F14 F3      OI          ; No interrupt
7F15 F5      PUSH   AF      ;
7F16 E5      PUSH   HL      ;
7F17 05      PUSH   DE      ; Save all
7F18 C5      PUSH   BC      ; registers to
7F19 00 E5    PUSH   IX      ; be used
7F1B F0 E5    PUSH   IY      ;
7F1D 3A EC 37 LD      A,(37ECH) ; Clear FDC
7F20 3A E0 37 LD      A,(37E0H) ; Clear F-F
7F23 21 69 7F LD      HL,MESSG ; Get "MEM="
7F26 11 36 3C LD      DE,VIDEO+36H ; Place to go
7F29 01 05 00 LD      BC,5      ; Its length
7F2C E0 8D    LOIR      ; Display it
7F2E AF      XOR      A      ; Clear carry
7F2F 2A E8 40 LD      HL,(STKBOT) ; BASIC stack
7F32 E0 58 FB 40 LD      DE,(ARRTOP) ; Array area
7F36 E0 52      SBC      HL,DE ; Room left
7F38 00 21 3B 3C LD      IX,VIDEO+3BH ; Place to go
7F3C F0 21 6E 7F LD      IY,TENTAB ; Tena-table
7F40 06 05      LD      B,5      ; No. digits
7F42 AF      XOR      A      ; Clear carry
7F43 F0 5E 00 LD      E,(IY+0) ; 10000 lobyte
7F46 F0 56 01 LD      D,(IY+1) ; 10000 hobyte
7F49 87      OR      A      ; Set flag
7F4A E0 52      SBC      HL,DE ; 1st diffnce.
7F4C 3B 03      JR      C,JUMPO ; If greater
7F4E 3C      INC      A      ; Additive div
7F4F 1B FB      JR      LOOP1 ; Divide again
7F51 19      JUMPO   ADD      HL,DE ; Was too much
7F52 C6 30      ADD      A,30H ; ASCII convrt.
7F54 00 77 00 LD      [IX+0],A ; Display it
7F57 00 23      INC      IX      ; Next screen
7F59 F0 23      INC      IY      ; Move once
7F5B F0 23      INC      IY      ; end again
7F5D 10 E3      DJNZ    LOOP0 ; Do 5 digits
7F5F F0 E1      POP      IY      ;
7F61 00 E1      POP      IX      ; Restore all
7F63 C1      POP      BC      ; the reg's
7F64 01      POP      DE      ; used to do
7F65 E1      POP      HL      ; the work
7F66 F1      POP      AF      ;
7F67 FB      EI          ;
7F68 CB      RET          ; Ints. okay
7F69 BF      DEFB      8FH ; BASIC awaits
7F6A          DEFM      'MEM=' ; Block
7F6A 40      7F6B 45      7F6C 40      7F6D 30
7F6E 10 27      TENTAB DEFW 100000 ; Tena-table
7F70 E8 03      DEFW 010000 ; is used for
7F72 64 00      DEFW 001000 ; division by
7F74 0A 00      DEFW 000100 ; subtraction
7F76 01 00      DEFW 000010 ;
7F00          END      SETUP

```

The 19-byte interrupt setup routine is found as usual, followed by the 100-byte service subroutine. All registers are saved — IY is optional but may be used by other utilities — a message is displayed, and a calculation is made to determine free memory. This calculation is the bottom of the BASIC subroutine stack (referenced at 40E8) minus the top of array space (referenced at 40FB).

Following that is a short, five-digit binary-to-ASCII table lookup which, in 35 bytes, converts and displays the free memory.

This routine is especially useful because it can help identify the actual area causing out-of-memory errors at run time. Heavily nested calculations can push the BASIC stack well down into memory, which is then visible on the screen. Put the program in place, and then try this :

```

10 GOSUB 20
20 GOTO 10

```

## Something Even More Useful

If you by chance have a printer which prints less than 40 characters per second, the following routine will speed up your BASIC programs. Otherwise, this routine will have little effect on your program speed (perhaps a two or three percent slowdown), but will allow a program to continue without 'locking up' while the printer is in action. It is called a 'spooler', a word which is based on an acronym mercifully forgotten.

When an LPRINT or LLIST is commanded, the spooler sends the characters to an in-memory buffer, and returns to the waiting program. Since 255 characters is the maximum string length, this buffer size is adequate; increasing it will be valuable only if the main program is highly interactive, involving user, screen, calculations and printer.

Normally, the LPRINT and LLIST routines send a character to the printer, and then check to see if the printer is 'busy'. In the meantime, the line printer routine is idle, performing no other work, and ignoring the BASIC program. Those of you with Teletypes know the discouraging feeling of waiting for hard copy.

Instead of waiting for the printer to output the entire number, word, or line before returning to the program, an interrupt service routine is used as a 'despooler'. When the 25-mS interrupt is generated, the despooler checks the 255-byte buffer to see if a printable character is present. If there is, the routine sends that character to the printer, advances the buffer, and returns to the program in progress. Also, if the printer is busy,

Listing 5-4. Print spooler.

```

00100 ; #####
00110 ; THIS INTERRUPT SERVICE ROUTINE STORES UP TO 255 CHAR-
00120 ; ACTERS IN A BUFFER, WHERE THEY MAY THEN BE OUTPUT TO
00130 ; A PRINTER. THIS SPOOLER CHECKS FOR CHARACTERS AT THE
00140 ; PRINTER DRIVER ROUTINE, INTERCEPTS THEM, STASHES THEM,
00150 ; AND RETURNS TO THE MAIN PROGRAM WITHOUT DELAY.
00160 ; #####
00170 ;
7E00      ORG      7E00H
00180
00190 ;
00200 ; #####
00210 ; SETUP ROUTINE CREATES A BUFFER AND PLACES ITS ADDRESS,
00220 ; ALONG WITH A JUMP TO REPLACE THE NORMAL RET, AT 4012H
00230 ; #####
00240 ;
7E00 F3      00250 6ETUP  OI          ; INT. OFF DURING SETUP
7E01 3EC3     00260 LD          A,0C3H ; GET JUMP VALUE INTO A
7E03 321240   00270 LD          [4012H],A ; PLACE INTO INT. VECTOR
7E06 AF       00280 XOR          A      ; CLEAR ACCUMULATOR
7E07 47       00290 LD          B,A    ; PUT 100 HEX INTO B
7E08 218A7E   00300 LD          HL,BUFFER-1 ; HL JUST AHEAD OF BUFFER
7E0B 23       00310 CLEAR  INC       HL ; NEXT POSITION IN BUFFER
7E0C 77       00320 LD          [HL],A ; PLACE ZERO INTO BUFFER
7E0D 10FC     00330 DJNZ       CLEAR ; 00 IT FULL 256 TIMES
7E0F 213F7E   00340 LD          HL,SERVE ; GET SERVICE ROUT. ADDR.
7E12 221340   00350 LD          [4013H],HL ; PLACE INTO INT. VECTOR
7E15 21237E   00360 LD          HL,SPOOL ; GET SPOOL ROUTINE
7E18 222640   00370 LD          [4026H],HL ; PLACE IN PRINTER DRIVER
7E1B 21191A   00380 LD          HL,1A1BH ; RETURN TO BASIC INTO HL
7E1E E5       00390 PUSH       HL    ; PLACE ON STACK
7E1F E056     00400 IM          1     ; SET INTERRUPT MODE
7E21 FB       00410 EI          ; INTERRUPTS READY TO GO
7E22 CB       00420 RET          ; BACK TO BASIC READY
00430 ;
00440 ; #####
00450 ; SPOOL ROUTINE STARTS HERE AND INTERCEPTS PRINTER DRIVER
00460 ; #####
00470 ;
7E23 F3      00480 SPOOL  OI          ; INT. OFF DURING SPOOL
7E24 E5       00490 PUSH       HL    ; SAVE HL REGISTER
7E25 7B       00500 LD          A,C    ; GET CHARACTER INTO A
7E26 F5       00510 PUSH       AF     ; SAVE CHAR. TO PRINT
7E27 218C7E   00520 LD          HL,BUFFER+1 ; POINT TO BUFFER START
7E2A FB       00530 EI          ; INTERRUPTS BACK ON NOW
7E2B 7E       00540 LOOP2  LD          A,[HL] ; GET BUFFER VALUE
7E2C A7       00550 AND          A      ; IS IT A CHAR. OR ZERO?
7E2D 20FC     00560 JR          NZ,LOOP2 ; WAIT IN LOOP IF FULL
7E2F F3       00570 OI          ; INTERRUPTS BACK OFF
7E30 06F0     00580 LD          B,0FDH ; GET PRESENT BUFFER SIZE
7E32 23       00590 LDP3   INC       HL    ; GET NEXT BUFFER POS'N
7E33 7E       00600 LD          A,[HL] ; BRING VALUE INTO A
7E34 A7       00610 AND          A      ; TEST FOR CHAR. OR ZERO
7E35 2002     00620 JR          NZ,SAVEIT ; FOUND FREE SPACE - GO
7E37 10FB     00630 DJNZ       LDP3   ; SEARCH THROUGH BUFFER
7E39 2B       00640 SAVEIT  DEC       HL    ; BACK OFF ONE POSITION
7E3A F1       00650 POP        AF     ; RESTORE CHAR. TO PRINT
7E3B 77       00660 LD          [HL],A ; PUT IT IN BUFFER
7E3C E1       00670 POP        HL    ; RESTORE FORMER VALUE
7E3D FB       00680 EI          ; INTERRUPTS BACK ON
7E3E C9       00690 RET          ; BACK TO MAIN ROUTINE
00700 ;
00710 ; #####
00720 ; INTERRUPT SERVICE ROUTINE FIRST SAVES REGISTERS, THEN
00730 ; RESETS INTERRUPT FLIP-FLOPS IN E/I. PRINTER STATUS
00740 ; IS EXAMINED, AND THE ROUTINE EXITED IF PRINTER BUSY.
00750 ; #####
00760 ;
7E3F F3      00770 SERVE   OI          ; INT. OFF DURING OESPOOL
7E40 F5       00780 PUSH       AF     ; SAVE VALUE IN ACCUM.
7E41 E5       00790 PUSH       HL    ; SAVE VALUE IN HL
7E42 05       00800 PUSH       OE     ; SAVE VALUE IN OE
7E43 C5       00810 PUSH       BC     ; SAVE VALUE IN BC
7E44 00E5     00820 PUSH       IX    ; SAVE VALUE IN IX
7E45 00212540 00830 LD          IX,4025H ; PRINTER CONTROL BLOCK
7E4A 3AEC37   00840 LD          A,[37ECH] ; RESET INT. FLIP-FLOP
7E4D 3AE037   00850 LD          A,[37E0H] ; RESET INT. FLIP-FLOP
7E50 3AEB37   00860 LD          A,[37EBH] ; GET PRINTER STATUS TO A
7E53 E6F0     00870 AND          0F0H ; MASK OUT LOW BITS
7E55 FE30     00880 CP          30H  ; SEE IF PRINTER IS BUSY
7E57 205A     00890 JR          NZ,OUT ; GO OUT IF PRINTER BUSY
00900 ;
00910 ; #####
00920 ; WHEN PRINTER IS READY, BUFFER IS MOVED UP, THE CHAR-
00930 ; ACTER IS TESTED, AND PRINTED IF A VALID CHARACTER.
00940 ; IF IT IS CARRIAGE RETURN, FORM FEED, LINE FEED, ETC.,
00950 ; APPROPRIATE TESTS ARE MADE IN THE PRINTER CONTROL BLOCK
00960 ; #####
00970 ;
7E59 21897F   00980 LD          HL,BUFFER+0FEH ; GET NEXT TO LAST CHAR.
7E5C 11BA7F   00990 LD          DE,BUFFER+OFFH ; GET NEXT CHAR. IN QUEUE
7E5F 01FF00   01000 LD          BC,OFFH ; GET TOTAL BUFFER SIZE
7E62 E0BB     01010 LDOR         ; MOVE IT UP ONE POS'N

```

Listing Continued . . .

it returns to the program immediately.

A ten-character-per-second (cps) printer like a Teletype is 'busy' for 100,000 microseconds for each character. In that time, a BASIC program might perform many calculations. Selectrics type a maximum of 15 cps, which is also time-consuming.

The program will not necessarily benefit in time with faster printers, but the waiting period will be eliminated in favor of a more interactive program overall.

The following listing is uncommented, as the bulk of the interrupt explanations have already been presented. The spooler is entered from BASIC via LLIST and LPRINT, either of which moves directly to the printer driver address referenced at 4026 and 4027. This is replaced with the address of the spooler. The character to be printed is delivered to the printer driver in the C register. The spooler searches through the buffer for the first non-zero value and places the character to be printed immediately before that. (If your system requires nulls after a carriage return, you can back up the appropriate number of places in the buffer).

Upon interrupt, the despooler saves registers and loads the IX register with 4025, the start of the printer device control block. This block contains line and page information, which is used to determine if paging has been completed, and how line feeds, carriage returns, and form feeds will be handled. It is functionally identical to the driver in ROM, except that it takes its characters from the spooler's buffer instead of directly from the Level II routines.

If the printer is not busy, then, the character is sent to the printer address at 37E8. Following is the complete routine, including setup, spool, and despool.

All of the interrupt programs presented in this article depend on the expansion interface for their 25 mS clock pulses. However, a simple rectified and shaped 16.66 mS pulse, which will work fine with all of these routines, can be created using the 60 Hz output of a 6.3-volt, low-current filament transformer, resistor, and 7414 Schmitt trigger.

The complete circuit for a printer decoder is available from Radio Shack as 'Printer Cable Service Manual'.

# Continued Listing

```

7E64 3ABA7F 01020 LD A,(BUFFER+OFFH) ; GET QUEUE CHAR TO PRINT
7E67 A7 01030 AND A ; TEST TO SEE IF ZERO
7E68 2B49 01040 JR Z,OUT ; GO OUT IF NONE LEFT
7E6A FE08 01050 CP 0BH ; CHECK IF TOP OF FORM
7E6C 280A 01060 JR Z,TEST1 ; TEST IF CHAR. IS ONE
7E6E FE0C 01070 CP 0CH ; TEST IF CHAR = F.F.
7E70 2D1F 01080 JR NZ,TEST2 ; NEXT TEST IF NOT
7E72 AF 01090 XOR A ; CLEAR ACCUMULATOR
7E73 008603 01100 OR [IX+3] ; GET LINES PRINTED
7E76 2B19 01110 JR Z,TEST2 ; OUT IF VALUE = 0
7E78 007E03 01120 TEST1 LD A,[IX+3] ; GET LINES PRINTED
7E7B 009604 01130 SUB [IX+4] ; GET LINES PER PAGE
7E7E 47 01140 LD 8,A ; PUT LINES LEFT IN B
7E7F 3AE837 01150 LOOPA LD A,[37E8H] ; GET PRINTER STATUS
7E82 E6F0 01160 AND 0F0H ; MASK OUT LOW 8 BITS
7E84 FE30 01170 CP 30H ; CHECK STATUS 8 BITS
7E86 20F7 01180 JR NZ,LOOPA ; LOOP IF PRINTER BUSY
7E88 3E0A 01180 LO A,DAH ; GET LINE FEED CHAR.
7E8A 32E837 01200 LO [37E8H],A ; SEND IT TO PRINTER
7E8D 10F0 01210 DJNZ LOOPA ; DO IT TILL FORM IS FED
7E8F 181E 01220 JR EXIT ; GO OUT WHEN DONE
7E91 47 01230 TEST2 LD 8,A ; STASH CHAR. IN 8 REG.
7E92 3AE837 01240 LOOPB LD A,[37E8H] ; GET PRINTER STATUS
7E95 E6F0 01250 AND 0F0H ; MASK OUT LOW 8 BITS
7E97 FE30 01260 CP 30H ; CHECK STATUS 8 BITS
7E99 20F7 01270 JR NZ,LOOPB ; LOOP TILL PRINTER READY
7E9B 78 01280 LO A,8 ; GET CHAR. BACK INTO A
7E9C 32E837 01290 LD [37E8H],A ; SEND IT TO PRINTER
7E9F FE00 01300 CP 00H ; CHECK IF CARRIAGE RET.
7EA1 2010 01310 JR NZ,OUT ; IF NOT THEN GO OUT
7EA3 003404 01320 INC [IX+4] ; ELSE INC. LINES COUNTER
7EA6 007E04 01330 LO A,[IX+4] ; GET NEW LINES COUNTER
7EA9 008E03 01340 CP [IX+3] ; CHECK WITH LINES/PAGE
7EAC 79 01350 LD A,C ; GET CHARACTER BACK TO A
7EAD 2004 01360 JR NZ,OUT ; GO IF NOT PAGE END
7EAF 00360400 01370 EXIT LO [IX+4],0 ; RESET PAGE LINES TO 0
01380 ;
01390 ; #####
01400 ; REGISTERS ARE RESTORED AND THE ROUTINE IS EXITED. THE
01410 ; LAST 255 BYTES OF SPACE ARE RESERVED FOR PRINTER BUFFER
01420 ; #####
01430 ;
7EB3 00E1 01440 OUT POP IX ; RESTORE IX REGISTER
7EB5 C1 01450 POP 8C ; RESTORE 8C REGISTER
7EB6 01 01460 POP 0E ; RESTORE 0E REGISTER
7EB7 E1 01470 POP HL ; RESTORE HL REGISTER
7EB8 F1 01480 POP AF ; RESTORE AF REGISTER
7EB9 F8 01490 EI ; INTERRUPTS BACK ON
7EBA C9 01500 RET ; BACK TO MAIN ROUTINE
00FF 01510 BUFFER DEFS 255 ; DEFINE 255-CHAR. BUFFER
7F8A 00 01520 DEF8 00 ; END BUFFER WITH 0 BYTE
01530 ;
01540 ; #####
7E00 01550 END SETUP
00000 TOTAL ERRORS
29420 TEXT AREA BYTES LEFT

BUFFER 7E88 01510 00300 00520 00980 00990 01020
CLEAR 7E08 00310 00330
EXIT 7EAF 01370 01220
LOOP2 7E28 00540 00560
LOOP3 7E32 00590 00630
LOOPA 7E7F 01150 01180 01210
LOOPB 7E92 01240 01270
OUT 7EB3 01440 00890 01040 01310 01360
SAVEIT 7E39 00640 00620
SERVE 7E3F 00770 00340
SETUP 7E00 00250 01550
SP00L 7E23 00480 00360
TEST1 7E78 01120 01060
TEST2 7E91 01230 01080 01110

```



## More Hardware Modifications

By now your TRS-80 is an electronic intimate. In this chapter, some significant (and some sophisticated) hardware modifications will be made. A few of these are simple, and their value is not immediately obvious. Others involve major work (such as the high-resolution graphics), but the results are exciting and very rewarding.

### Making Halt Work

Let's start with an easy one. The HALT instruction is a useful command. It isn't exceptional, but you will find that time spent idling in loops waiting for an interrupt is easier when the HALT instruction is used. In fact, its main advantage comes when the Z-80 microprocessor is used in interrupt-based systems. Where program work is being done, the HALT instruction is not particularly meaningful; but where program operations are suspended except for interrupts, it is very useful.

When HALT is executed on the Z-80, the processor simply ceases program counter operation and executes NOPs in order to continue memory refresh. That is, the Z-80 outputs a fetch (the M1 or machine cycle one signal) which – together with a refresh address on the address bus – constitutes enough information to keep the memory active. The information received during the fetch is ignored by the CPU, which continues to execute NOPs until an

interrupt is received. It then exits the NOP state and services the interrupt.

Whenever the HALT instruction is executed, a halt-acknowledge signal (HALT, active low) is output on pin 18. In the TRS-80, pin 18 is tied to an input of NAND gate Z53. The other input of Z53 is tied high through a 10K resistor, but can be pulled low when the Reset button is pressed. In other words, either a Reset or a HALT will redirect the CPU to the address of the non-maskable interrupt (NMI), where it will continue execution from either 'READY' or DOS reboot.

By cutting loose pin 18 of the CPU (Z40), and tying Z53 pin 2 high, with a 4.7K ohm resistor, the HALT instruction will operate as intended by the Z-80's designers. Memory will continue to be refreshed, as the 4116s' refresh cycle depends on RAS (row address strobe), which will be output as usual by Z72.

Cut the trace running from Z40, pin 18. Attach a 4.7K ohm resistor from Z53 pin 2, to Z53 pin 14. The modification is complete.

### A Real Break

Later in this Chapter is a power-on monitor, which will give more importance to the minor change described here. The TRS-80 does not have a true 'break' function that resets the CPU to its power-up condition. It can be simulated by entering SYSTEM followed by /0, or by pressing the Reset button on a disk system.

This change is quite simple. Locate Z53, a 74LS132 NAND gate, which feeds (through Z52) the system reset pin of the Z-80 microprocessor. When the power is turned on, capacitor C42 takes time to charge, holding the processor in its reset mode for a few milliseconds. When the capacitor charges completely, the input to Z53 is held high by resistor R47.

Run a wire from pins 12 and 13 of Z53, through a 1K resistor, to one end of a SPST (single pole single throw) normally open pushbutton switch. The other end of the switch goes to ground (which can be found at Z53, pin 7). When this switch is pressed, the Z53 sees a low signal, and resets the processor to address 0000. A resistor is used in series so that C42 is not overexerted by shorting directly to ground.

Be sure to mount this button well out of the way; for disk systems, it's as fatal as the Reset button. For Level II it means MEMORY SIZE? and the loss of any program in memory.

## Stuck Relays

There are two ways to fix a stuck relay: bang it until it unsticks, or replace it. The many published fixes to tape recorders, or additional external circuitry, simply don't take into consideration any changes you might make to your tape system; nor to the possibility you may take your CPU with you when you work elsewhere.

A replacement relay is available from *Lab Service, Inc., Box 383, Hustisford, Wisconsin 53034*. Unlike the relay normally installed in the TRS-80, it is reliable, low power and has gold contacts. I have installed this unit in my computer, and run the high current motor of a CTR-41 through 100,000 operations continuously over the course of a week. Neither the CTR-41 nor the relay failed.

The cassette relay is mounted just behind the jacks for power, video and cassette. It is a cylinder approximately the size of the LSI replacement relay, but in most TRS-80s it contains six leads instead of four.

Using solder-wick and a hot soldering iron, heat the solder connections and draw off excess solder. While doing this, slide a thin flat blade under the relay. Take care not to move the nearby small video trimmer potentiometers. Use the blade as a lever, lifting the old relay off the board. Apply very gentle pressure to this lever, and alternately melt the three solder connections on each side of the relay. Do not use force that

might crack the board or lift solder traces from the board.

When one side of the relay is completely free, grasp it with your hand and pull *gently*, while heating the remaining three connections on the opposite side of the relay. It will eventually pull free.

Remove excess solder from the holes with the solder-wick. Use a fine splinter to open the eyes of the holes, if all the solder does not flow into the solder-wick.

Now examine the circuit board, noting that on the end of the relay position nearest the computer's connection jacks there are two otherwise unused connections; no circuit traces run out from these points. Look at the replacement relay, noting that it has four wires, three on one side and one on the other. Orient it above the board so that the unused holes match the end of the relay with the single wire. This wire feeds into the center of the trio of holes.

Carefully insert the relay in place; if you have properly cleaned out the connection holes, the new relay wires will slide in, barely protruding from the opposite side of the board. You may have to bend them just a bit in order to get them to feed through.

Once the wires are in place, apply a very small amount of solder, and secure the relay on the board. Check carefully for solder splashes, shorts or cracks, and clean or repair them. Such a check is doubly important here because the relay driver circuit (unlike most other circuits in the TRS-80) cannot handle a short circuit for very long.

Refit the boards and covers of the computer together, replace the power and cassette cables; then put the cassette player into its play position, and enter this program:

```
10 PRINT#-1," "
20 FOR N = 1 TO 500 : NEXT
30 GOTO 10
```

This program will turn the cassette player on and off at regular intervals. If it does not work properly, double check all connections, especially the orientation of the relay, and try again. This fix should eliminate any further concern over a sticking relay causing skipped data or missed program loads.

## High Resolution Graphics

Now for the biggie. In the past few years, there has been quite a bit of excitement generated by the idea of high-resolution graphics. Reasonably representative images can be drawn with them, and animation is considerably more exciting; especially when compared with the extremely high resolution of the type used in the latest generation of coin-operated video games. The Apple computer was advertised with a heavy emphasis on their high resolution, hard as it is to work with. The new Radio Shack Color Computer also offers several modes of resolution.

The following project can either stand alone as a plug-in peripheral, or be integrated as part of the TRS-80 keyboard unit. In either case, the specifications are:

1. Resolution of 384 dots wide by 192 dots deep.
2. Full compatability with all current software.
3. Simultaneous use *and overprint* of normal TRS-80 alphanumerics *and graphics*.
4. Addressing using six bits in contiguous memory blocks of 768 bytes each; sixteen total memory blocks are used.

The hardware involved in this project, including power supply and miscellaneous hardware, will be under \$100 (probably closer to \$70 by the time you read this), yet will compete easily with any high-resolution add-on for the TRS-80.

On the negative side, this project will involve a great deal of wire wrapping or soldering, and will eat up one chunk of 16K memory address space when it is used. It will not actually compete with or replace the top memory block in the expansion box (there is no electronic conflict) but will be addressed from C000 to FFFF. Alternatively, it may be addressed from 8000 to BFFF. In either case, you do not need the expansion box to run this memory.

The TRS-80 screen has 1,024 locations in a grid of 64 characters across by 16 lines down. Within each of these grid elements are blocks 2

pixels by 3 pixels (a pixel is a 'picture element') for the familiar coarse graphics mode accessed with SET and RESET. If you turn the contrast fully down and reduce the brightness of the screen, the individual dots which make up the graphic and alphanumeric characters can be seen with a sharp eye; a magnifying glass will make the dots very clear.

In order to produce a complete screen line of letters, the locations in video memory are handed to a circuit which actually accesses them twelve times – once for each pass the electron beam makes horizontally. At each pass, a row of dots corresponding to part of the whole line of letters is shifted out to the video beam. Each dot (or 'undot') then turns the beam on or off for the tiny fraction of a second it takes to sweep across 1/384th of the screen.

The point is clear: an electronic event takes place for every dot on every line of the screen. This means that it is possible to create an individual, addressable electronic event for each screen dot.

The process might work like this:

1. Devise a circuit whose timing characteristics are identical to the video timing of the usual TRS-80 circuits.
2. Instead of addressing the same video memory on twelve consecutive screen lines, have the addressing select *different* memory for each line.
3. Have the contents of that memory filled by the TRS-80, and displayed on the same or a different monitor. The add-on has its own video circuitry, but can be displayed on the same monitor because step 1 specifies that the timing characteristics must be identical to those in the TRS-80. It's like an auto with 4-wheel drive, where all wheels are capable of working together; or a dual-capstan tape recorder, where both capstans pull the tape to ensure steady contact with the playback head.

The circuit shown opposite presents the complete high-resolution circuitry. There are two ways of building this circuit since the areas shaded in grey *are already present within the TRS-80*. If you wish, you can solder directly to those circuits inside the computer, saving yourself some parts and perhaps a bit of time. Otherwise, the entire circuit can be built separately.

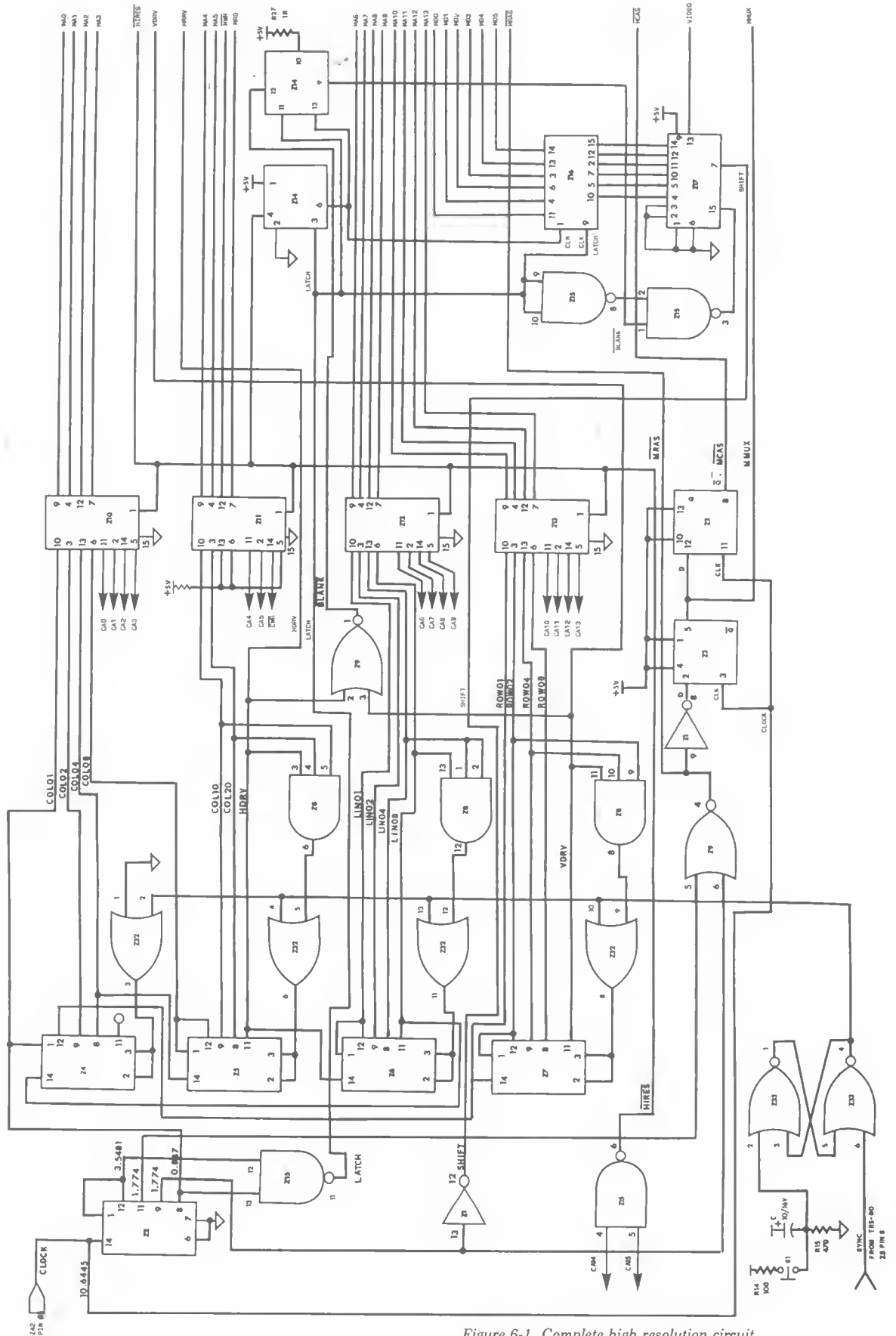


Figure 6-1. Complete high resolution circuit.

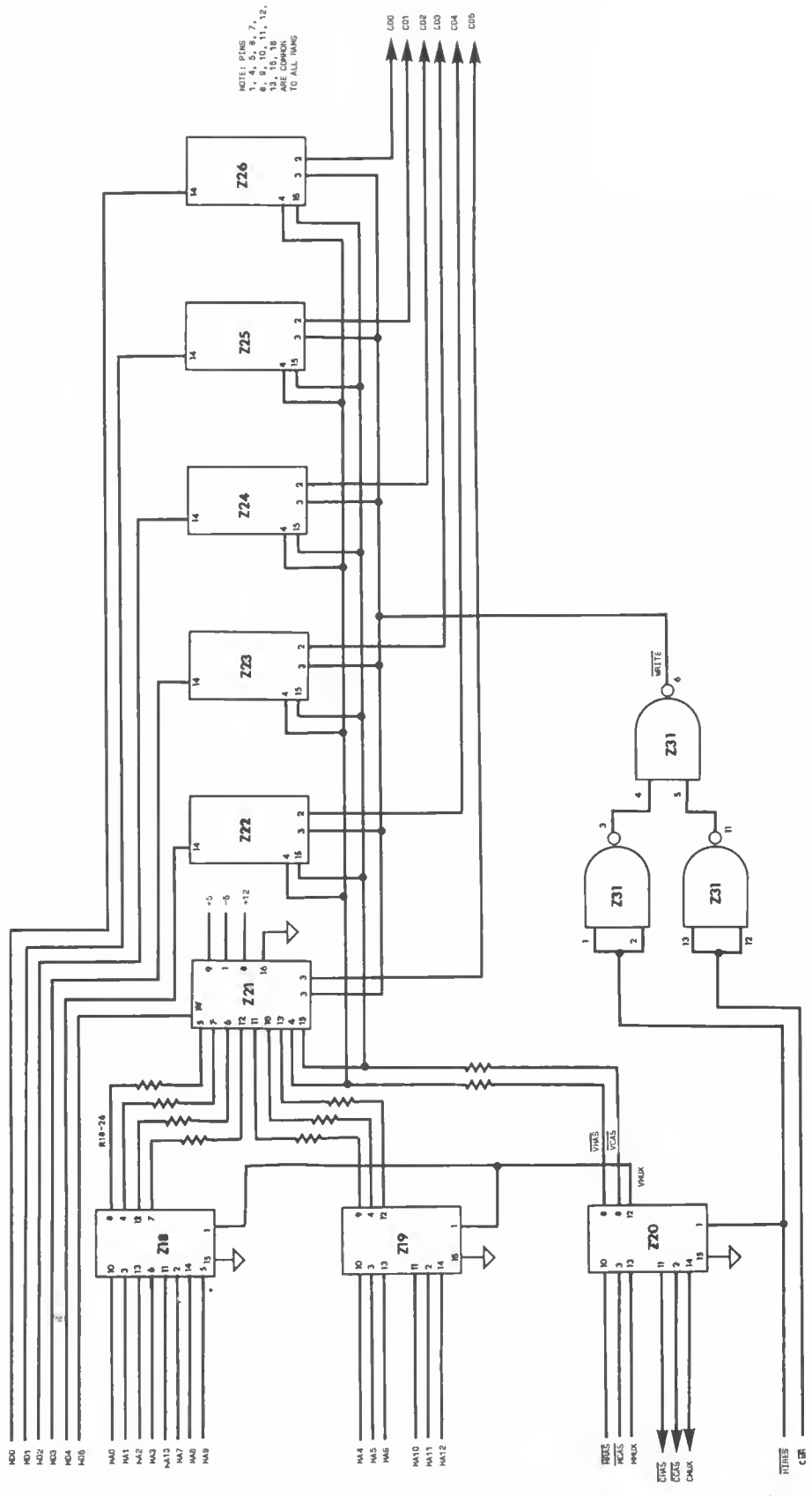


Figure 6-1b. High resolution memory.

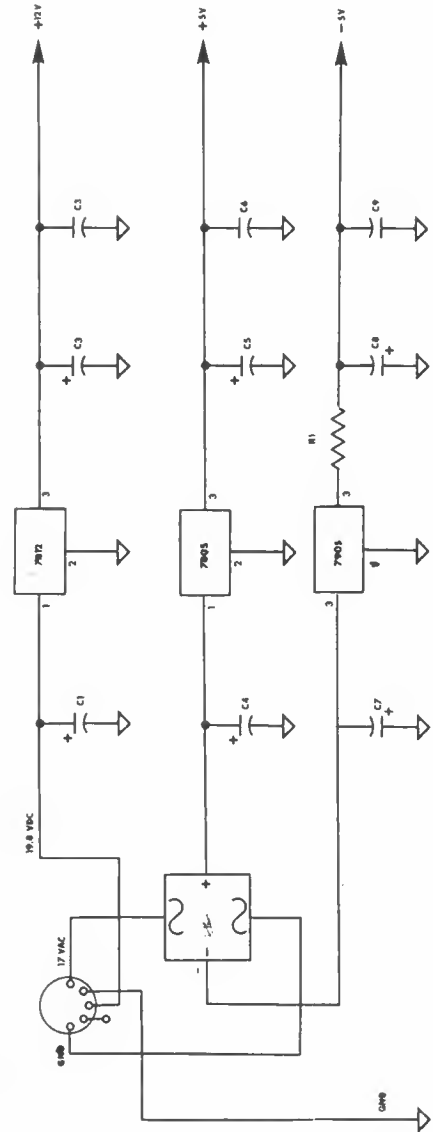


Figure 6-1c. High resolution power supply.

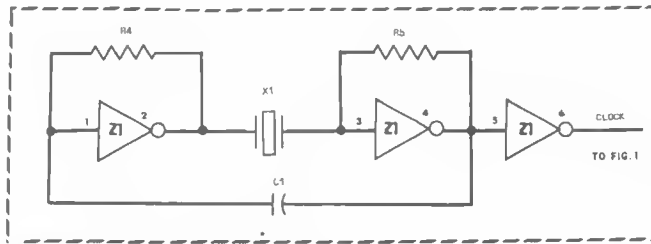


Figure 6-2. Hires clock circuit.

Figure 6-2 is the clock. The 10.6445 MHz crystal is the same one used in the TRS-80; Radio Shack sells it for \$3.95, on special order. A small trimmer capacitor is included so that the frequency of the high resolution board can be aligned identically to that in the computer.

Figure 6-3 is the master video countdown chain and timing circuitry, also nearly identical to that inside the TRS-80. There are a few exceptions. The logic necessary for 32 characters per line is not present (it is not needed in high resolution mode, although the normal alphanumeric may be displayed in 32-character mode simultaneously with the hi-res graphics screen). Also, the four outputs of Z6 do not feed any latches or character generators; instead, they become the line of dots addresses for high resolution memory. Identical (top and side) screen blanking is used.

Figure 6-1. is the high-resolution memory itself. The familiar 4116 type, 16K dynamic memories are used in this circuit (250 nS or less is essential), but with a difference. The hi-res board must generate its own memory refresh, yet hand over control to the TRS-80 when it needs to select memory into which it will write information. Thus, Z20 multiplexes the on-board refresh/select (MRAS, MCAS, MMUX) with the TRS-80 select (CRAS, CCAS, CMUX).

The switch is made by the simultaneous presence of addresses 14 and 15 on the address bus.

On-board refresh/select is generated by the clock in combination with two flip-flops (Z3a/b), producing select in this order:

1. Z2 pin 8, selects the lowest portion of the address; as such, it is the fastest changing memory select signal.
2. Two clock cycles later, Z2 pin 11, produces a signal which will be gated by Z9 and inverted to produce MRAS (row address strobe).
3. One clock cycle later, MMUX goes high, produced by the clocking of Z3a.
4. One clock cycle later, MRAS is continued as Z2 pin 11 goes low by Z2 pin 9 going high. The transition is simultaneous and virtually invisible.
5. At that time, MCAS is produced when Z3b is clocked low at the NOT Q output.
6. Memory data is stable at this time, and two clock cycles later, LATCH is issued by Z1e, latching the data into Z16 for its trip through shift register Z17.

Figure 6-3 shows the creation of horizontal and vertical synchronization signals, and the horizontal and vertical screen positioning controls. This circuitry again is identical to that in the TRS-80, as is Figure 6-4, the video mixing circuitry.

There is only one critical construction area in the device, and that is the circuitry surrounding the 10.6445 MHz crystal (Z16, R4-5, C1). The wires in this area must be very short, and all the parts clustered together. Capacitor C1 should be the only part of the circuit responsible for tuning the crystal's frequency, not random capacitance

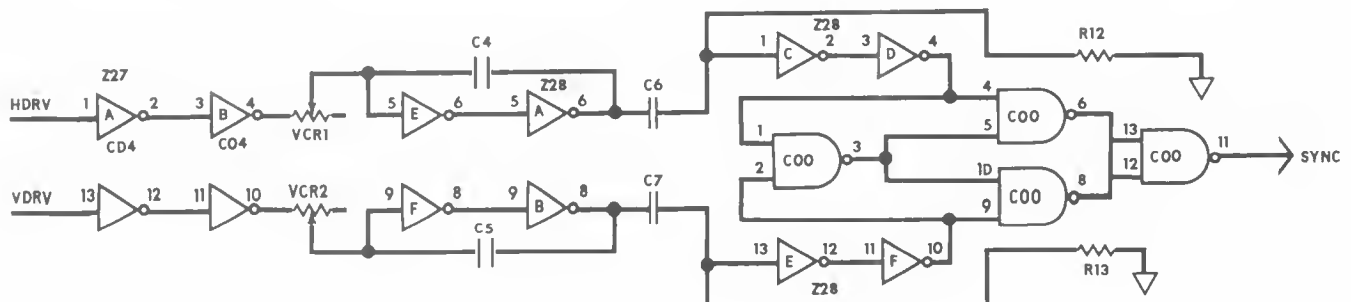


Figure 6-3. Master video circuit.

and graphics with the high-resolution dots.

To use the device, attach a 5-pin DIN cable between the TRS-80 video jack and the input jack hi-res board. Connect the video monitor to the output jack of the hi-res board, then attach a 40-pin edge connector from the TRS-80 to the hi-res board. Turn the mixing control fully counterclockwise (Hi-Res Out). Power up the hi-res board, and then the rest of the system in normal order.

As usual, MEMORY SIZE? should appear; if so enter 49152 (for a 48K computer). The system should operate as usual. Enter the following program:

```
10 FOR X = 15360 TO 16383
20 POKE X,129
30 NEXT
40 FOR X = -16384 TO 0
50 POKE X,175
60 NEXT
70 GOTO 70
```

The screen will fill with small graphics blocks. There will be a pause of almost a minute while the rest of the program is running. Put an AM radio next to the computer to determine when the program is complete. Now bring the mixing control of the hi-res board clockwise until dots, herringbone, jitter and/or other interference appears on the screen. This is a good sign.

If you have a stable enough screen to see the alternating dot patterns produced by the hi-res board, then adjust the horizontal and vertical positioning controls, if necessary, to center the image with that of the TRS-80. To remove the jitter and herringbone adjust the fine tuning control.

Now your screen should display an alternating pattern of TRS-80 graphics, along with an overlay of thin vertical lines of hi-res graphics dots. If you have any difficulty getting this pattern, or if there are any other problems, refer to the troubleshooting section.

Use of the Hi-Res board is simple. Addresses from C000 through C03F contain the information to create the first line of dots, addresses C040 through C07F contain the second line, etc. A contiguous block of memory from C000 through C2FF is used for the first twelve lines of dots. But since the display is twelve lines, and not sixteen, the addressing takes a jump in order to be compatible with the familiar 64 x 16 normal screen display. Thus, addresses C300 through C3FF are not used, and the second group of 12 dot lines begins at address C400 and continues

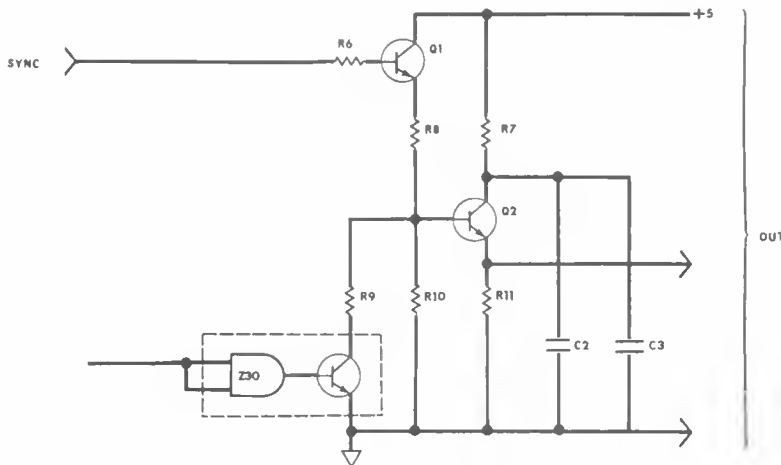


Figure 6-4. The video mixing circuitry.

introduced by a haphazard bunch of wires.

Other than the wire layout near the crystal, construction is time-consuming but straightforward. I recommend wire wrapping the entire circuit; use different colors for data, address, video, ground, etc., so that troubleshooting will be simplified.

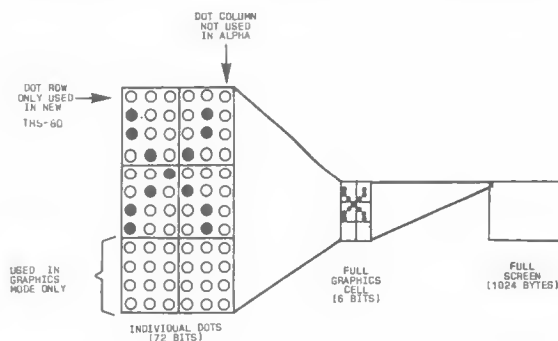
The completed circuit will have these external controls:

1. **Power.** Three voltages, +5, -5, and +2 are necessary. -5 volts must be present first and last.
2. **Mixing.** This controls the intensity of the high-resolution board with respect to the TRS-80 alphanumerics and graphics.
3. **Fine tuning.** This adjusts the frequency of the 10.6445 MHz crystal to that of the TRS-80. Occasional adjustments will be necessary with temperature changes.
4. **Vertical and horizontal positioning.** These control the placement of the image on the screen; it should coincide with the alphanumeric screen normally produced by the TRS-80.
5. **Input.** This accepts a cable running from the TRS-80 video jack, which would normally attach to the video monitor.
6. **Output.** This accepts a cable from the video monitor, and provides an output which mixes the TRS-80 alphanumerics

through C6FF. Here is a full memory map of the Hi-Res board:

Hi-Res Board Memory Map		
C000 - C03F	Group 1, Line 1, Screen Line 1	
C040 - C07F	Group 1, Line 2, Screen Line 2	
C080 - C0BF	Group 1, Line 3, Screen Line 3	
C0C0 - C0FF	Group 1, Line 4, Screen Line 4	
C100 - C13F	Group 1, Line 5, Screen Line 5	
C140 - C17F	Group 1, Line 6, Screen Line 6	
C180 - C1BF	Group 1, Line 7, Screen Line 7	
C1C0 - C1FF	Group 1, Line 8, Screen Line 8	
C200 - C23F	Group 1, Line 9, Screen Line 9	
C240 - C27F	Group 1, Line 10, Screen Line 10	
C280 - C2BF	Group 1, Line 11, Screen Line 11	
C2C0 - C2FF	Group 1, Line 12, Screen Line 12	
C400 - C43F	Group 2, Line 1, Screen Line 13	
C440 - C47F	Group 2, Line 2, Screen Line 14	
C680 - C6BF	Group 2, Line 11, Screen Line 23	
C6C0 - C6FF	Group 2, Line 12, Screen Line 24	
C800 - CAFF	Group 3, Lines 1 - 12 Screen Lines 25 - 36	
CC00 - CEFF	Group 4, Lines 1 - 12 Screen Lines 37 - 48	
D000 - D2FF	Group 5, Lines 1 - 12 Screen Lines 49 - 60	
D400 - D6FF	Group 6, Lines 1 - 12 Screen Lines 61 - 72	
D800 - DAFF	Group 7, Lines 1 - 12 Screen Lines 73 - 84	
DC00 - DEFF	Group 8, Lines 1 - 12 Screen Lines 85 - 96	
E000 - E2FF	Group 9, Lines 1 - 12 Screen Lines 97 - 108	
E400 - E6FF	Group 10, Lines 1 - 12 Screen Lines 109 - 120	
EB00 - EAFF	Group 11, Lines 1 - 12 Screen Lines 121 - 132	
EC00 - EEFF	Group 12, Lines 1 - 12 Screen Lines 133 - 144	
FD00 - F2FF	Group 13, Lines 1 - 12 Screen Lines 145 - 156	
F400 - F6FF	Group 14, Lines 1 - 12 Screen Lines 157 - 168	
FB00 - FAFF	Group 15, Lines 1 - 12 Screen Lines 169 - 180	
FC00 - FEFF	Group 16, Lines 1 - 12 Screen Lines 181 - 192	

Only six bits of each byte are used (the least significant six); thus, six one-bit-wide, memory chips are used in the circuit. The bits fit into their respective lines and memory addresses as follows:



Before continuing, clear out the hi-res memory with:

```
FOR X = -16384 TO 0 : POKE X,0 : NEXT
```

Drawing simple lines is an easy process; for a horizontal one, just enter:

```
FOR X = -12288 TO -12224 : POKE X, 63 : NEXT
```

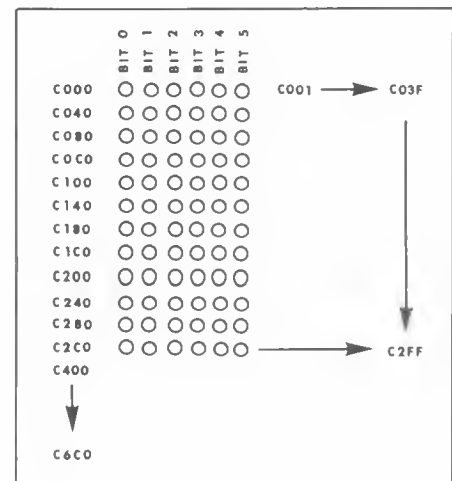
A vertical one is produced by stepping through groups:

```
5 CLS : REM KILL ALPHANUMERICS
10 FOR Y = -16352 TO 0 STEP 1024 : REM STEP THROUGH GROUPS
20 FOR X = Y TO Y+(12*64) STEP 64 : REM STEP THROUGH LINES
30 POKE X,1 : NEXT X : REM SET ONE PIXEL
40 NEXT Y : REM TO NEXT LINE GROUP
50 GOTO 50 : REM KEEP DISPLAY INTACT
```

Diagonal lines are more complicated, because more than two sets of increments must be specified; but simple diagonals can be created. For diagonals and variable-width lines, change listing to read as follows:

```
10 INPUT Q : INPUT R : CLS
20 FOR Y = -16352 TO 0 STEP 1024
30 FOR X = Y TO Y+(12*64) STEP 64+Q
40 POKE X,(R AND 63) : NEXT X
50 NEXT Y
60 GOTO 60
```

For serious graphics, assembly language programming is the only way real speed can be achieved. This is a very 'custom' type of programming, and only the simplest of subroutines will be presented here. For drawing circles, ellipses, and curves the functions will have to be stored in a look-up table and calculated. Listing 6-1 is an assembly listing to draw graphic lines on the screen, given a set of coordinates.



```

00100 ; #####
00110 ; ROUTINE TO DRAW HORIZONTAL AND VERTICAL LINES FR. BASIC
00120 ; #####
00130 ;
C000 00140 HIRES EQU 0C000H ; START OF HIRES GRAFIX
0A7F 00150 XFER EQU 0A7FH ; VARIABLE XFER ROUTINE
00160 ;
7F00 00170 ORG 7F00H ; SOMEWHERE IN MEMORY
00180 ;
00190 ; #####
00200 ; SUBROUTINE TO DETERMINE THE CORRECT BASIC USR(X) CALL
00210 ; #####
00220 ;
7F00 C07F0A 00230 ENTRY CALL XFER ; GET VALUE FROM BASIC
7F03 7C 00240 LO A,H ; GET MSB INTO ACCUM.
7F04 B5 00250 ADD A,L ; ADD LSB FROM HL PAIR
7F05 A7 00260 AND A ; TEST IF IT IS ZERO
7F06 2B0B 00270 JR Z,PCLS ; CLEAR SCREEN ROUTINE
7F0B FE01 00280 CP 1 ; TEST IF IT IS ONE
7F0A 2B15 00290 JR Z,PHORIZ ; HORIZONTAL LINE ON 1
7F0C FE02 00300 CP 2 ; TEST IF IT IS A TWO
7F0E 2B30 00310 JR Z,PVERT ; VERTICAL LINE ON 2
7F10 C39719 00320 JP 1997H ; SN? ERROR IF NOT 1,2,3
00330 ;
00340 ; #####
00350 ; SUBROUTINE TO CLEAR THE SCREEN IN HIGH-RESOLUTION MODE
00360 ; BASIC FORMAT: M=USR(A). A MUST ALWAYS BE ZERO.
00370 ; #####
00380 ;
7F13 AF 00390 PCLS XOR A ; GET CHARACTER TO WRITE
7F14 FS 00400 PUSH AF ; SAVE THAT CHARACTER
7F15 2100C0 00410 LD HL,HIRES ; GET BEGINNING OF HI-RES
7F1B F1 00420 PCLEAR POP AF ; RESTORE ZERO CHARACTER
7F19 77 00430 LO (HL),A ; PUT IT IN PLACE IN MEM
7F1A FS 00440 PUSH AF ; SAVE CHARACTER AGAIN
7F1B 7C 00450 LO A,H ; GET MSB OF CURRENT LOC.
7F1C B5 00460 OR L ; GET LSB AND TEST PAIR
7F10 20F9 00470 JR NZ,PCLEAR ; LOOP BACK FOR NEXT
7F1F F1 00480 POP AF ; GET STACK BACK IN SHAPE
7F20 C9 00490 RET ; BACK TO CALLING ROUTINE
00500 ;
00510 ; #####
00520 ; SUBROUTINE TO DRAW A HORIZONTAL LINE. BASIC FORMAT:
00530 ; POKEN,B:C1=INT(C/6):C2=C-C1*6:POKEN+1,C1:POKEN+2,C2:
00540 ; O1=INT(O/6):O2=O-O1*6:POKEN+3,O1:POKEN+4,O2:M=USR(1)
00550 ;
00560 ; B=LINE NUMBER (0-11, 16-27, 32-43, 48-59, 64-75, 80-91,
00570 ; 96-107, 112-123, 128-139, 144-155, 160-171, 176-
00580 ; 187, 192-203, 208-219, 224-235, 240-251
00590 ; C=ORIGIN POSITION (0-383).
00600 ; O=DESTINATION POSITION (0-383). C MUST BE LARGER THAN O
00610 ; #####
00620 ;
7F21 0021B97F 00630 PHORIZ LO IX,N ; POINT TO THE STORAGE
7F25 C06E7F 00640 CALL FINDER ; GET STARTING POSITION
7F2B 3AB7F 00650 LD A,(N+3) ; GET VALUE AT "N+3"
7F2B 4F 00660 LO C,A ; PLACE IN MSB OF BC
7F2C AF 00670 XOR A ; CLEAR ACCUM TO ZERO
7F20 47 00680 LO B,A ; PLACE IN LSB OF BC
7F2E ES 00690 PUSH HL ; SAVE START LOC'N
7F2F 05 00700 PUSH OE ; READY FOR XFER BACK
7F30 E1 00710 POP HL ; TRANSFERRED BACK
7F31 09 00720 ADD HL,BC ; PERFORM THE ADDITION
7F32 ES 00730 PUSH HL ; READY FOR TRANSFER
7F33 01 00740 POP OE ; AND GET INTO DESTIN'N
7F34 E1 00750 POP HL ; RESTORE ORIGINAL VALUE
7F35 3AB7F 00760 LD A,(N+2) ; GET VALUE AT "N+2"
7F3B B6 00770 OR (HL) ; ADD TO CURRENT LOC'N
7F39 77 00780 LO (HL),A ; AND PUT INTO PLACE
7F3A 23 00790 INC HL ; NEXT SCREEN POSITION
7F3B AF 00800 LOOP1 XOR A ; CLEAR ACCUM TO ZERO
7F3C 3A3F00 00810 LD A,(03FH) ; VALUE TO FILL BYTE
7F3F E052 00820 SBC HL,OE ; PERFORM SUBTRACTION
7F41 2B04 00830 JR Z,HOROUT ; OUT OF ROUTINE
7F43 77 00840 LO (HL),A ; PUT BYTE IN PLACE
7F44 23 00850 INC HL ; GET NEXT SCREEN POS'N
7F45 1B4 00860 JR LOOP1 ; GO BACK FOR NEXT FILL
7F47 3AB07F 00870 HOROUT LO A,(N+4) ; GET BACK FINAL BYTE
7F4A B6 00880 OR (HL) ; ADD TO VALUE ON SCREEN
7F4B 77 00890 LD (HL),A ; PUT IT ON THE SCREEN
7F4C C9 00900 RET ; AND BACK TO BASIC
00910 ;
00920 ; #####
00930 ; SUBROUTINE TO DRAW A VERTICAL LINE. 48K RAM IN PLACE.
00940 ; BASIC FORMAT:
00950 ; POKEN,B:C1=INT(C/6):C2=C-C1*6:POKEN+1,C1:POKEN+2,C2:
00960 ; O1=INT(O/6):O2=O-O1*6:POKEN+3,O1:POKEN+4,O2:M=USR(2)
00970 ;
00980 ; E=HORIZONTAL POSITION (0-255)
00990 ; F=LINE NUMBER (SEE HORIZONTAL LINE, ABOVE)
01000 ; G=LINE NUMBER (SEE HORIZONTAL LINE, ABOVE)
01010 ; #####
01020 ;
7F40 0021B97F 01030 PVERT LO IX,N ; POINT TO THE STORAGE

```

Listing 6-1. Hires demonstration program.

Listing Continued . . .

Without TRS-80 memory parallel to that in the hi-res board, it is not possible to read the contents of the high-resolution memory directly.

The contents must be stored in some form elsewhere. When the high 16K block is in place in the expansion box, however, six bits of each byte are identical to those on the screen. Ideally, the entire block of high resolution memory (16K by 6 bits) and TRS-80 memory (16K by 8 bits) should be cleared out by POKEing 0 in place first. Then an in-computer image of the high resolution screen can be maintained at all times.

Another interesting mode of using the high resolution board is with a separate screen. Normal alphanumerics can appear on the TRS-80 monitor, while the high-resolution graphics can be presented on a parallel screen. This way, the table calculations and information reported can be displayed on the computer's monitor for reference. The high-resolution screen will be unaffected by anything done by the TRS-80 unless its memory is being specifically written to. Not only can action games of the Startrek type be more interesting and challenging — with visuals and info displayed on different screens — but for experimentation and analysis, the high-resolution display is unbothered by program changes.

To use this mode, merely leave your TRS-80 monitor plugged into the computer. Then send the video information in the hi-res board to another video monitor, or to an ordinary television set via an RF modulator.

## Troubleshooting

With a complicated project like this, there is a good chance that the system will not work perfectly the first time. The main problems together with their causes and solutions are outlined below.

1. The screen keeps tearing or jittering no matter what setting the fine tuning is on. If the fine tuning has no effect at all, it may be defective. Replace it. If the tuning gets better, but can't quite pull it in, you can put another capacitor in parallel to increase the capacitance, or replace the crystal (in either the hi-res board or the TRS-80) with one better matched to the other.
2. The high-resolution graphics cannot be changed, remaining the same as when the power was turned on. The memory write circuits are not working properly. Check the memory-select wiring at Z20; the write

## Troubleshooting

### Continued Listing

```

7F51 C06E7F 01040 CALL FINDER ; GET SCREEN START BYTE
7F54 E5 01050 PUSH HL ; SAVE THE START VALUE
7F55 00218C7F 01060 LD IX,N+3 ; POINT TO THE TABLE
7F59 C06E7F 01070 CALL FINDER ; AND DO THE WORK
7F5C E5 01080 PUSH HL ; READY VALUE TO TRANSFER
7F5D 01 01090 POP DE ; AND TRANSFER TO DEST'N
7F5E E1 01100 POP HL ; RESTORE START POSITION
7F5F 014000 01110 LD BC,40H ; SCREEN BYTE LINE OFFSET
7F62 AF 01120 XOR A ; CLEAR CARRY FLAG
7F63 3AB87F 01130 LD A,(N+2) ; GET BYTE FROM STORAGE
7F66 77 01140 LD (HL),A ; STASH IT ON SCREEN
7F67 09 01150 ADD HL,BC ; MOVE UP ON THE SCREEN
7F68 E052 01160 SBC HL,DE ; CHECK IF DONE YET
7F6A 20FA 01170 JR NZ,LOOP2 ; BACK IF NOT DONE
7F6C 77 01180 LD (HL),A ; PUT LAST BYTE IN
7F6D C9 01190 RET ; BACK TO BASIC
01200 ;
01210 ; #####
01220 ; FINDER SUBROUTINE LOCATES PROPER BYTE WITHIN HIRES AREA
01230 ; #####
01240 ;
7F6E 2100C0 01250 FINDER LD HL,HIRES ; GET HI-RESOLUTION SCRIN
7F71 D07E00 01260 LD A,(IX) ; GET START BYTE OFFSET
7F74 47 01270 LO B,A ; PLACE IN B REGISTER
7F75 AF 01280 XOR A ; CLEAR THE CARRY FLAG
7F7B C81B 01290 RR B ; DIVIDE BY TWO AND ...
7F7B C81B 01300 RR C ; ... ROTATE IN ORDER ...
7F7A C81B 01310 RR B ; ... ACTUALLY TO ...
7F7C C81B 01320 RR C ; ... MULTIPLY BY 64
7F7E 08 01330 ADD HL,BC ; MAKE NEW SCREEN POS'N
7F7F E5 01340 PUSH HL ; READY IT FOR TRANSFER
7F80 01 01350 POP DE ; TRANSFER TO DESTIN'N
7F81 D07E01 01360 LD A,(IX+1) ; GET START BIT OFFSET
7F84 4F 01370 LD C,A ; PLACE IN MSB OF BC
7F85 AF 01380 XOR A ; CLEAR ACCUM TO ZERO
7F8B 47 01390 LD B,A ; PLACE IN LSB OF BC
7F87 09 01400 ADD HL,BC ; GET NEW START BIT
7F8B C8 01410 RET ; BACK TO CALLING ROUTINE
01420 ;
7F89 01430 N EQU $ ; MEMORY STORAGE POS'NS
01440 ;
06CC 01450 END 06CCH ; BACK TO BASIC
00000 TOTAL ERRORS
30026 TEXT AREA BYTES LEFT

ENTRY 7F00 00230
FINDER 7F8E 01250 00640 01040 01070
HIRES C000 00140 00410 01250
HOROUT 7F47 00870 00830
LOOP1 7F38 00800 00860
LOOP2 7F66 01140 01170
N 7F8B 01430 00630 00650 00760 00870 01030 01080 01130
PCLEAR 7F1B 00420 00470
PCLS 7F13 00390 00270
PHORIZ 7F21 00630 00290
PVERT 7F4D 01030 00310
XFER 0A7F 00150 00230

```

lines to the memory (Z11); and the write line from the computer (from edge card pin 13).

3. The high-resolution graphics keep changing without writing to them. The memory-select circuits may be selecting write for both read and write; check Z11. More likely, the memory refresh/select circuitry is miswired; check Z9a and Z3.

4. When creating lines of graphics, the dots do not appear in the correct place. This indicates the memory data and /or address lines are miswired; check the lines from the computer (pins 4-7, 9-11, 17-18, 20, 22, 24-28, 30-32, 34-36, 38 and 40), making sure they are in the correct order. Also check each memory circuit to be sure the address lines (pins 4-7, 10-13, and 15) are parallel in each memory IC. Finally, be sure Z4 correctly feeds Z10; Z5 correctly feeds Z11; Z6 correctly feeds Z12; and Z7 correctly feeds Z13. These four circuits are the memory count/multiplex circuits. Also check that Z10, 11, 12 and 13 correctly feed Z18 and 19.

5. No graphics are produced. This is a tough one. The fault could lie with

- (a) the clock formed by Z1a-c
- (b) the memory refresh circuits Z9b and Z3
- (c) latch and shift registers Z16 and Z17
- (d) memory circuits Z21 to Z26
- (e) video output formed by Z30, Q1 and Q2.

Check the screen display carefully, because if any of these sections are working (except the video output) the screen will be affected in some way, even if it is minor. If herringbone or some tearing is present when the fine tuning is adjusted, then the video output and sync circuits are probably okay. Also, be sure that the mixing control is not turned fully counterclockwise (TRS-80 on, hi-res off).

6. The computer crashes to MEMORY SIZE? or otherwise acts problematically. The hi-res board has no effect on the computer. No data is written to the computer from the hi-res board at any point; it only receives information. If the computer crashes, then faulty wiring is likely.

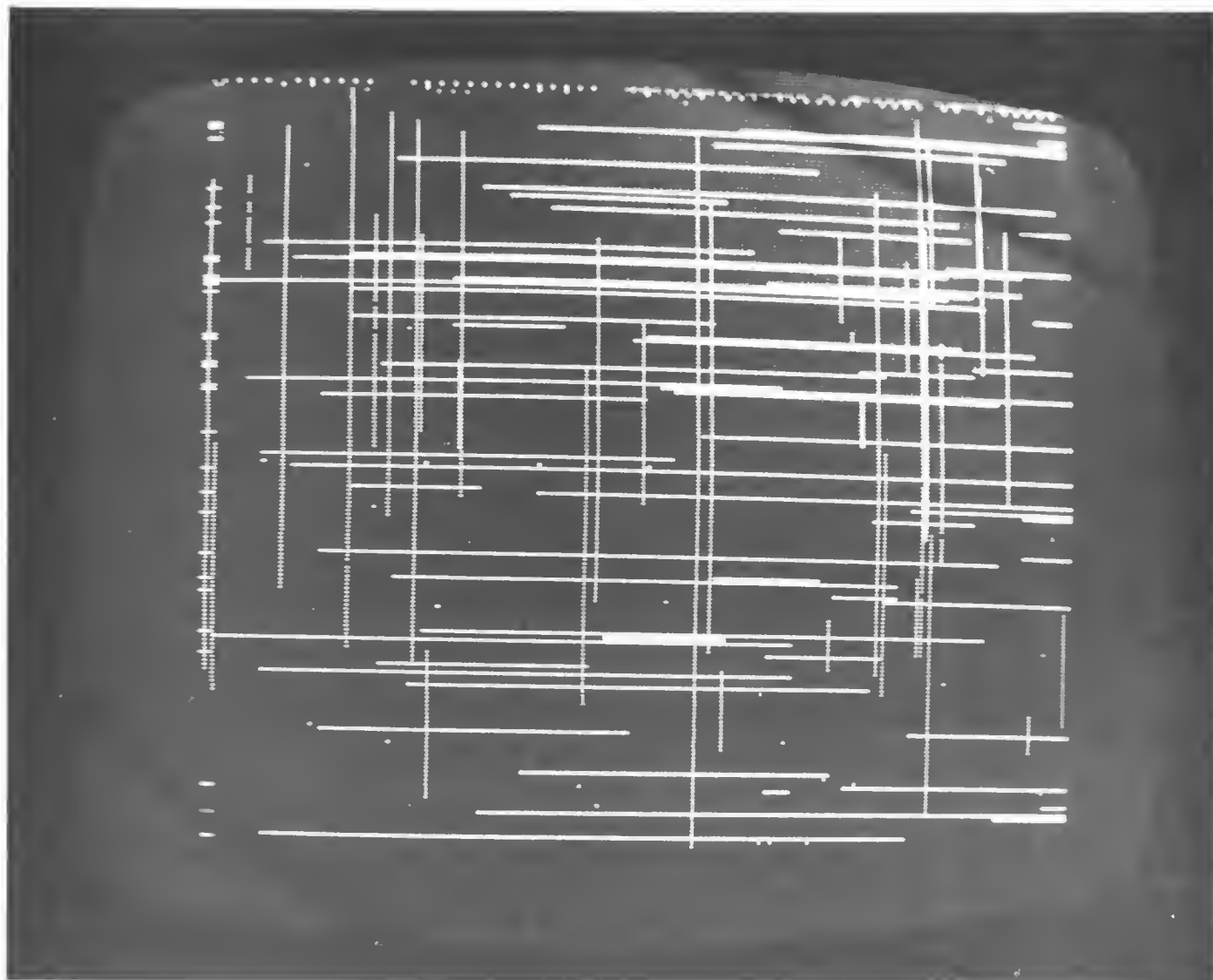


Photo 6-1. Hi-Res graphics example.

## Replace BASIC ROMs

In the past few years, more and more TRS-80 users have asked me the question, "I don't need BASIC because I do all my work in machine language. How can I use that space for my own routines and language in ROM?". The answer is simple: replace your ROMs with 2716 (2K x 1 byte), 2732 (4K x 1 byte) or 2764 (8K by 1 byte) EPROMs.

The process is electronically simple. The Level II (or Level I) ROMs are removed, and a board containing the new *and old* ROMs is inserted at the edge connector. This board, then, can select either ROMs at the flick of a switch – exactly like the Apple's 'softcards' do it.

This section will present a circuit to use 2716 EPROMs and any other ROMs together and, as a bonus, a way to hand control of your TRS-80 over to another *microprocessor*! The power of such processors as the 6502, 6800 series, the 8060 (SC/MP) and others then becomes available to the TRS-80 user. Together with the appropriate bootstrap and executive programs in ROM, the TRS-80 can act with the strengths of almost any language and almost any processor.

There is no secret to adding ROM. The area from 0000 to 37C0 is free to use, and some of that method has already been presented. Figure 6-5 presents the circuitry that will handle seven 2716 EPROMs, switchable with the 3-chip Level II ROM set.

The real trick is making another processor available to the TRS-80's hardware. This other processor must be able to:

- access memory and peripherals in the TRS-80 memory map
- address at least 32K of memory
- be able to move its software stack anywhere in memory
- refresh the 4116 dynamic memories.

Of these restrictions, the last two are the trickiest. Without also placing RAM on this outboard device, the 6502 cannot be used, because it requires its stack in page 1 (0100-01FF) and many of its fastest (and hence most advantageous) instructions are limited to page 0 (0000-00FF).

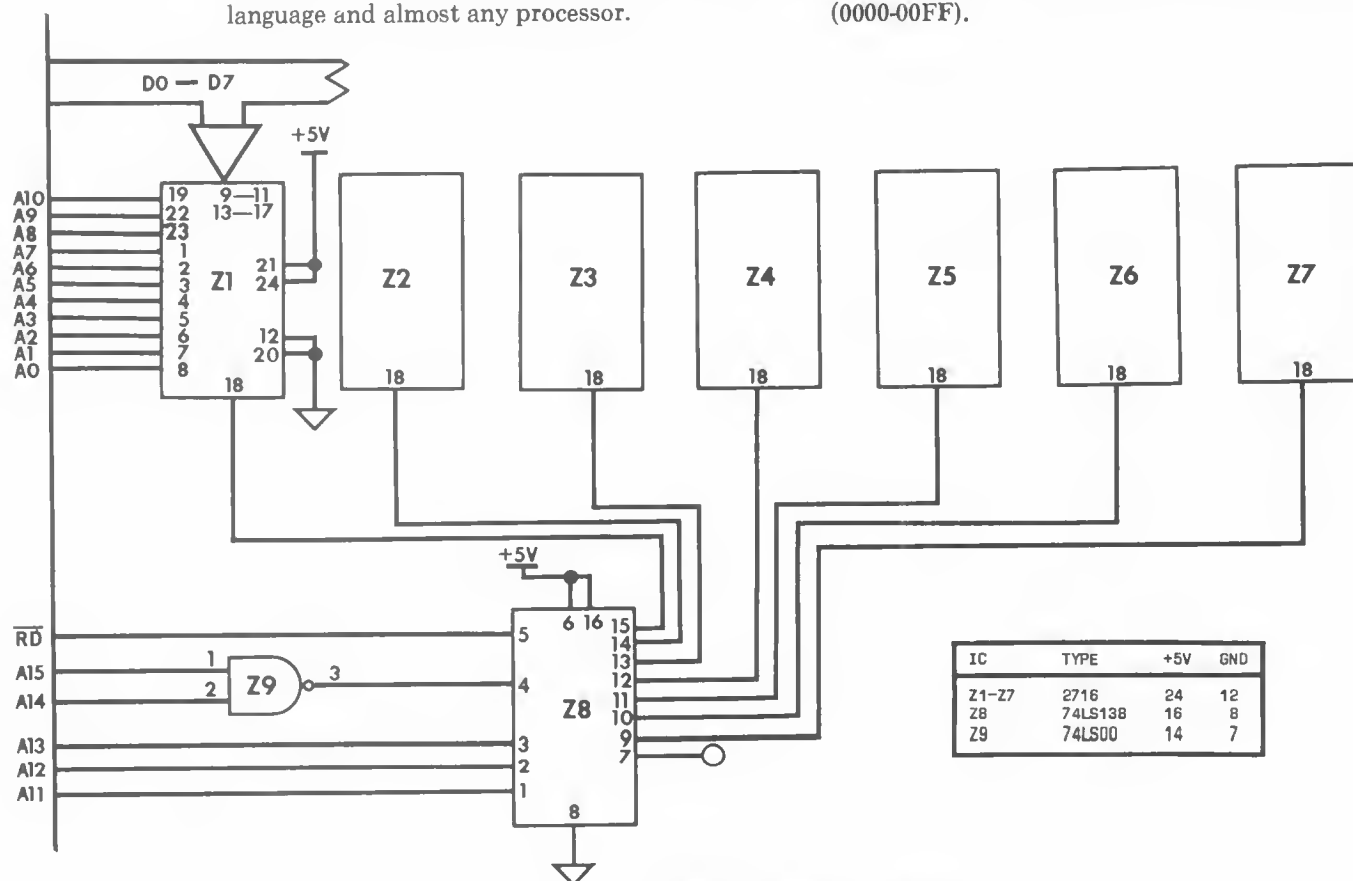


Figure 6-5. Circuit for replacing BASIC ROMs.

1. REMOVE DIP SHUNT Z3.
2. ALL SEVEN 2716'S ARE CONNECTED IN PARALLEL EXCEPT PIN 18.

However, the refresh requirement is the most severe. Because the RAS-only refresh is controlled within the TRS-80, and its signals are killed when the processor is removed, the entire refresh process must be handled outside the computer by chips which were not designed for dynamic memory support.

Before turning to the details, you may be asking "How can control be wrested from the Z-

80?". The answer is found in the line marked TEST. This input line is present on pin 23 of the TRS-80 bus, and when brought low, causes all address lines (A0-A15), all data lines (D0-D7), read (RD), write (WR), input (IN), output (OUT), row address strobe (RAS), column address strobe (CAS) and memory select multiplex (MUX) to be put into three-state condition. This provides a wide-open computer bus for the outside world.

## Power-Up Monitor

From my programmer (but not my user!) point of view, entering BASIC immediately upon power-up, or reset, is inconvenient and sometimes maddening. Rather, I would prefer a jump to a monitor of the type which can be found on the Ohio Scientific microcomputers: instead of MEMORY SIZE?, the user is presented with 'D/C/W/M?', which means 'Disk reboot, Cold start in BASIC (MEMORY SIZE?), Warm start in BASIC (READY), or Machine Language Monitor?'.

The user can exit from BASIC to the monitor and back at any point without jumping to a program-creating MEMORY SIZE? The cold start serves that purpose, and is almost always by choice.

There is such a possibility on the TRS-80, so long as two conditions are met:

1. A machine language monitor is resident, preferably in ROM.
2. The restart from 0000 is redirected from BASIC to that machine language monitor, and/or the NMI located at 0066 is redirected to that machine language monitor. This latter configuration preserves the last location of the program counter on the stack for examination.

The first requirement is easy to meet, and is in my opinion the most logical use of a ROM memory addition.

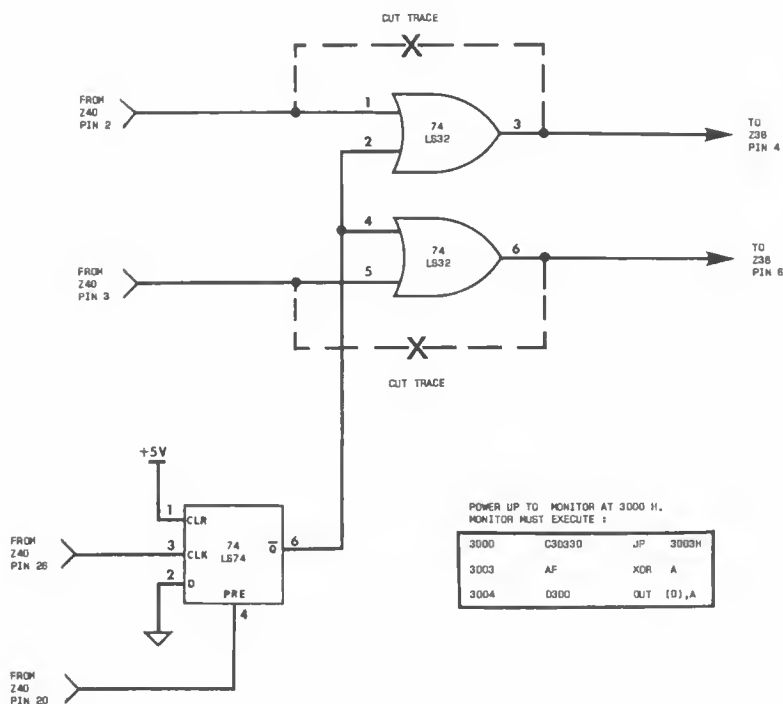


Figure 6-6. Modification needed for power-up monitor.

```

00100 ; #####
00110 ; POWER-UP PROGRAM FOR USE INDEPENDENTLY (WITHOUT BASIC
00120 ; ROMS IN PLACE) OR IN CONJUNCTION WITH EXIT AND RE-ENTRY
00130 ; TO BASIC UP RESET TO 0000 (BEFORE DISC SYSTEM REBOOT).
00140 ; #####
00150 ;
401A 00160 KPLACE EQU 401AH ; RAM STORAGE FOR STROKE
3000 00170 ORG 3000H ; MONITOR ENTRY POINT
00180 ;
00190 ; #####
00200 ; OPTIONAL SCREEN-CLEAR. IF THE EXIT CONDITION OF THE
00210 ; SCREEN IS IMPORTANT, THEN ELIMINATE CLEARING PROCESS
00220 ; #####
00230 ;
3000 F3 00240 OI ; KILL THE INTERRUPTIONS
3001 E073E242 00250 LD [42E2H],SP ; SAVE STACK POINTER
3005 31E042 00260 LD SP,42E0H ; GET NEW STACK POINTER
3008 F5 00270 PUSH AF ; SAVE REGISTERS FOR
3009 C5 00280 PUSH BC ; LATER DISPLAY
300A 05 00290 PUSH DE ; DURING THE
300B E5 00300 PUSH HL ; MONITOR ENTRY
300C 21003C 00310 CLS LO HL,3C00H ; BEGINNING OF SCREEN
300F 11013C 00320 LO OE,3C01H ; NEXT POSITION ON SCREEN
3012 01FF03 00330 LD BC,3FFH ; NUMBER OF SCREEN POS'NS
3015 3620 00340 LD (HL),20H ; PRINT SPACE 1ST POS'N
3017 E0B0 00350 LOIR ; DO IT FOR WHOLE SCREEN
00360 ;
00370 ; #####
00380 ; DISPLAY DISK/MONITOR/WARM START/USER ROUTINE MESSAGE
00390 ; #####
00400 ;
3019 218732 00410 LD HL,MS601 ; GET FIRST MESSAGE
301C 11C03C 00420 LD OE,3CC0H ; GET DISPLAY POSITION
301F C0B32 00430 CALL DISPLY ; AND DISPLAY ON SCREEN
3022 CDC331 00440 MENU CALL INPUT ; GET INPUT FROM KEYBOARD
3025 FE63 00450 CP 63H ; IS IT A LETTER "C"?
3027 2B13 00460 JR Z,C0LO ; GO TO C0LOSTART ROUTINE
3028 FE64 00470 CP 64H ; IS IT A LETTER "O"?
302B CA0000 00480 JP Z,0000 ; GO TO DISK REBOOT
302E FE60 00490 CP 60H ; IS IT A LETTER "M"?
3030 2B30 00500 JR Z,MONTOR ; GO TO MACHINE MONITOR
3032 FE75 00510 CP 75H ; IS IT A LETTER "U"?
3034 2B22 00520 JR Z,USER ; GO TO USER ROUTINE
3036 FE77 00530 CP 77H ; IS IT A LETTER "W"?
303B 2B2A 00540 JR Z,WARM ; GO TO BASIC WARMSTART
303A 1BE6 00550 JR MENU ; LOOP BACK IF NONE
00560 ;
00570 ; #####
00580 ; C0LOSTART ROUTINE DUPLICATES LEVEL II ACTION TO 0075H.
00590 ; STACK POINTER NOT RESET HERE BECAUSE C0LOSTART DOES IT.
00600 ; #####
00610 ;
303C F3 00620 C0LO OI ; INTERRUPTS TURNED OFF
303D AF 00630 XOR A ; CLEAR ACCUMULATOR
303E 03FF 00640 CLEAR OUT [0FFH],A ; BEGIN WRITE-PORT LOOP
3040 210206 00650 LD HL,0602H ; GET COMMUNICATION BLOCK
3043 110D40 00660 LD OE,4000H ; GET COMMUNICATION AREA
3046 013600 00670 LD 8C,0036H ; GET SIZE OF COMM. BLOCK
3049 E0B0 00680 LDIR ; BLOCK MOVE COMM. AREA
304B 30 00690 DEC A ; A = A-1 (A=0 AT START)
304C 30 00700 DEC A ; A = A-1 (FE, FC, FO...)
304D 20EF 00710 JR NZ,CLEAR ; CLEAR ALL EVEN PORTS
304F 0627 00720 LD B,27H ; GET 39 VAR. LOCATIONS
3051 12 00730 LOOP1 LD [0E],A ; BEGIN SETTING UP CHARS
3052 13 00740 INC OE ; GET NEXT MEM POS'N
3053 10FC 00750 DJNZ LOOP1 ; LOOP THROUGH THEM ALL
3055 C37500 00760 JP 0075H ; REST OF BASIC STARTUP
00770 ;
00780 ; #####
00790 ; USER ENTRY ROUTINE REQUIRES THAT A TWO-BYTE ADDRESS B
00800 ; STORED IN MEMORY AT 408E/40BF (16526/16527), <=USR(0)>
00810 ; #####
00820 ;
3058 E1 00830 USER POP HL ; BEGIN TO CLEAR THE
3059 01 00840 POP OE ; STACK OF ALL
305A C1 00850 POP BC ; REGISTERS PUSHED
305B F1 00860 POP AF ; AT ENTRY POINT
305C E07BE242 00870 LD SP,(42E2H) ; GET ORIGINAL STACK PTR
3060 2ABE40 00880 LD HL,(40BEH) ; GET USR(0) ENTRY POINT
3063 E9 00890 JP (HL) ; AND JUMP TO IT
00900 ;
00910 ; #####
00920 ; WARM START ROUTINE RESTORES STACK BEFORE GOING TO BASIC
00930 ; #####
00940 ;
3064 E1 00950 WARM POP HL ; RESTORE ALL REGISTERS
3065 01 00960 POP OE ; PUSHED ON STACK
3066 C1 00970 POP BC ; WHEN MONITOR
3067 F1 00980 POP AF ; WAS ENTERED
306B E07BE242 00990 LD SP,(42E2H) ; GET ORIGINAL STACK PTR
306C C3CC06 01000 JP 06CCH ; GO TO BASIC WARMSTART
01010 ;

```

Listing 6-2. Power-up monitor program.

## How Interpreters Work

The concept of computer languages is far too complex for a single section, a single chapter or even a single book. They run from easy, messy, but capable languages like BASIC through hard and messy languages like FORTRAN, to neat, structured, but obtuse languages like Pascal and LISP. In between are pseudo BASICs like BASEX, plus a whole range of compiled languages and hybrid self-definers like FORTH.

All these languages have one thing in common: they must eventually be broken into subroutines which operate in the machine language of the host processor. Compiled languages are broken into these subroutines when the program is written, but the bulk of material installed in small computers is accessed most easily by means of interpreters – executing one statement at a time, during a program's run.

How does a computer get to the information to be interpreted or compiled? Here's a short rundown of how an interpreter does its work; compilers are similar, but they won't be covered here.

Once a program has been constructed (and the method varies from machine to machine) it is in place for the computer to evaluate and execute. The process that follows is consistent with most interpreters:

Once a program has been constructed, and the method varies from machine to machine, it is in place for the computer to evaluate and execute. The process that follows is consistent with most interpreters:

1. Upon an execution command, the interpreter identifies the start of the program.
2. The first command from the program is obtained. The interpreter compares this command, byte for byte, against a table of legitimate commands. In some interpreters, the commands are stored as full words; in others, they are tokenized. When a valid entry is found in the command table, this stage of interpretation is complete. If a valid entry is not found, the routine is exited, usually via a loop which can be intercepted by extensions to the interpreter.

```

01020 ; #####
01030 ; MACHINE LANGUAGE MONITOR IS SIMPLE, DISPLAYING REGISTER
01040 ; CONDITIONS AT ENTRY, AND ACCEPTING MEMORY CHANGES. ALL
01050 ; REGISTERS ARE RESTORED UPON EXIT FROM THIS ROUTINE.
01060 ; #####
01070 ;
306F 210632 01080 MONITOR LD HL,MSG02 ; GET REGISTER DISPLAY
3072 11C03C 01090 LD OE,3CC0H ; GET SCREEN POSITION
3075 C06F32 01100 CALL DISPLY ; DISPLAY THE MESSAGE
3078 110030 01110 LD OE,3000H ; GET SCREEN POSITION
3078 ED48E242 01120 LD BC,(42E2H) ; GET STACK POINTER POS'N
307F C5 01130 PUSH BC ; GET VALUE READY TO XFER
3080 E1 01140 POP HL ; TRANSFER BYTES TO HL
3081 C01731 01150 CALL WORDER ; CONVERT & DISPLAY BYTE
3084 00E5 01160 PUSH IX ; GET IT READY TO USE
3086 E1 01170 POP HL ; AND DO IT WITH IT
3087 C01731 01180 CALL WORDER ; CONVERT & DISPLAY BYTES
308A F0E5 01180 PUSH IY ; GET IT READY TO USE
308C E1 01200 POP HL ; AND DO IT WITH IT
3080 C01731 01210 CALL WORDER ; CONVERT & DISPLAY IT
3080 2ADE42 01220 LD HL,(42DEH) ; POINT TO AF STACK VALUE
3083 C01731 01230 CALL WORDER ; CONVERT & DISPLAY BYTE
3086 2ADC42 01240 LD HL,(42DCH) ; POINT TO BC STACK VALUE
3089 C01731 01250 CALL WORDER ; CONVERT & DISPLAY BYTE
308C 2A0A42 01260 LD HL,(42DAH) ; POINT TO DE STACK VALUE
309F C01731 01270 CALL WORDER ; CONVERT & DISPLAY BYTE
30A2 2AD842 01280 LD HL,(42D8H) ; POINT TO HL STACK VALUE
30A5 C01731 01290 CALL WORDER ; CONVERT & DISPLAY BYTE
30A8 E05F 01300 LD A,R ; GET R REGISTER INTO A
30AA 67 01310 LD H,A ; GIVE I FOR CONVERSION
30AB E057 01320 LD A,I ; GET I REGISTER INTO A
30AD 6F 01330 LD L,A ; GIVE IT FOR CONVERSION
30AE C01731 01340 CALL WORDER ; CONVERT & DISPLAY BYTE
30B1 118030 01350 LD OE,3080H ; GET NEW VIDEO POSITION
30B4 08 01360 EX AF,AF' ; TRANSFER OTHER VALUE
30B5 F5 01370 PUSH AF ; READY TO XFER TO HL
30B6 E1 01380 POP HL ; EFFECT TRANSFER TO HL
30B7 08 01390 EX AF,AF' ; RETURN ORIGINAL VALUE
30B8 C01731 01400 CALL WORDER ; CONVERT & DISPLAY BYTE
30B8 09 01410 EXX ; TRANSFER BC, DE, HL
30BC E5 01420 PUSH HL ; FIRST SLIP BC ON STACK,
30BD 05 01430 PUSH DE ; THEN PUSH DE THERE,
30BE C5 01440 PUSH BC ; AND THEN THE LAST ONE.
30BF 09 01450 EXX ; TRANSFER REGISTERS BACK
30C0 E1 01460 POP HL ; GET VALUE ALL READY
30C1 C01731 01470 CALL WORDER ; CONVERT & DISPLAY BYTE
30C4 E1 01480 POP HL ; GET DE VALUE READY
30C5 C01731 01490 CALL WORDER ; CONVERT & DISPLAY BYTE
30C8 E1 01500 POP HL ; GET HL VALUE READY
30C8 C01731 01510 CALL WORDER ; CONVERT & DISPLAY BYTE
30CC 11003F 01520 CHECK LD OE,3F00H ; GET NEW SCREEN POS'N
30CF C01731 01530 CALL WORDER ; CONVERT & DISPLAY BYTE
30D2 1C 01540 INC E ; BUMP SCREEN POSITION
30D3 7E 01550 LD A,(HL) ; GET VALUE FROM MEMORY
30D4 C02431 01560 CALL HEXASC ; AND DISPLAY IT IN ASCII
01570 ;
01580 ; #####
01590 ; ADDRESS MOD ROUTINE GETS KEYBOARD VALUE & LOOPS BACK
01600 ; #####
01610 ;
3007 CDC331 01620 ADDRMOD CALL INPUT ; GET VALUE FROM KEYBOARD
300A FE21 01630 CP 'I' ; TEST FIRST IF EXECUTE
300C 2832 01640 JR Z,EXEC ; OUT IF EXECUTE COMMAND
300E FE2F 01650 CP '/' ; TEST SECOND IF DATA
30E0 CA6E31 01660 JP Z,0ATMOD ; OUT IF DATA MOD COMMAND
30E3 FE2A 01670 CP '*' ; TEST THIRDO IF RET MENU
30E5 CA0C30 01680 JP Z,CLS ; OUT TO MENU IF A STAR
30E8 FE30 01690 CP '0' ; SEE IF <0 CHARACTER
30EA 38E8 01700 JR C,ADDRMOD ; LOOP BACK IF <0 CHAR.
30EC FE67 01710 CP 67H ; SEE IF >F CHARACTER
30EE 30E7 01720 JR NC,ADDRMOD ; LOOP BACK IF >F CHAR.
30F0 FE3A 01730 CP ':' ; SEE IF <9 CHARACTER
30F2 3804 01740 JR C,NUMBER ; GO TO NUMBER ROUTINE
30F4 FE61 01750 CP 61H ; SEE IF >A CHARACTER
30F6 38DF 01760 JR C,ADDRMOD ; LOOP BACK IF <A & >9
30F8 21043F 01770 NUMBER LD HL,3F04H ; GET ADDRESS SCREEN POSN
30F8 11033F 01780 LD OE,3F03H ; GET NEXT SCREEN POS'N
30FE 010300 01790 LD 8C,3 ; GET THREE TOTAL POS'NS
3101 E080 01800 LDIR ; AND MOVE THEM OVER
3103 28 01810 DEC HL ; REPOSITION TO LAST CHAR
3104 FE60 01820 CP 60H ; COMPARE TO L.C. ALPHA
3106 3802 01830 JR C,ZIPBY ; IF NUMERIC, THEN SKIP
3108 0620 01840 SUB 20H ; ELSE CONVERT TO U.CASE
310A 77 01850 ZIPBY LD (HL),A ; DISPLAY NEW CHARACTER
3108 C04331 01860 NUM2 CALL ASCHEX ; CONVERT DISPLAY TO HL
310E 188C 01870 JR CHECK ; AND LOOP BACK FOR MORE
3110 21063F 01880 EXEC LD HL,3F06H ; POINT HL TO ADDRESS
3113 C04331 01890 CALL ASCHEX ; AND CONVERT TO HEX
3116 E9 01900 JP (HL) ; HL CONTAINS ADDRESS
01910 ;
01920 ; #####
01930 ; SUBROUTINE RESPONSIBLE FOR CONVERTING & DISPLAYING TWO-
01940 ; BYTE WORDS DELIVERED TO IT IN THE HL REGISTER.

```

Listing Continued . . .

3. If the command is identified as a legitimate one, a subroutine is called which executes the command. That subroutine in turn may further examine the command line for operands and conditions, incrementing and decrementing pointers in its search for required, and valid, information. Further subroutines are entered as necessary to evaluate and put to use this additional information. In-memory variables and pointers are set up, modified, and accessed by all subroutines, usually from a master table which defines variable types and syntax conditions. Transcendental functions are also accessed via tables within the interpreter itself. Error checking is done at all points.

4. When the execution of each subroutine is completed, it returns to the calling program. Eventually, all subroutines have returned to their upper level of subroutine 'nesting'. Then the execution routine finds itself re-entered, positioned at the next executable point in the program, where the execution process is repeated.

5. The execution routine may, depending on the language, find itself repositioned in the program out of normal execution sequence. On the other hand, some languages have an inherent structure which disallows any repositioning, demanding an inviolable first-to-last execution sequence. In these latter interpreters, repositioning will be interpreted as an error condition.

6. If any information, commands, program order or program syntax is incorrect, an error handling routine is entered, usually by direct jump rather than subroutine call. Normally, this routine prepares and presents an error message. It returns the program from execution level, arbitrarily cancelling any nested execution subroutines by readjusting stack pointers and other variable information.

This readjustment is necessary to avoid unsettling the user-interactive command-level routine, and causing the processor to lose its way in a complex of incomplete subroutines.

7. Upon completion of all program statements, the program is returned from execution level to command level. Some interpreters allow commands to be entered, interpreted, and executed from a command buffer, without actually entering a program execution condition.

## Continued Listing

```

01950 ; #####
01960 ;
3117 1C 01870 WORDER INC E ; BUMP OE REGISTER ALONG
3118 1C 01980 INC E ; SOME MORE BUMPING ...
3119 1C 01990 INC E ; ... AND SOME MORE OF IT
311A 7C 02000 LD A,H ; GET FIRST BYTE OF IT
311B C02431 02010 CALL HEXASC ; CONVERT & DISPLAY BYTE
311E 7D 02020 LD A,L ; GET SECOND BYTE OF IT
311F C02431 02030 CALL HEXASC ; CONVERT & DISPLAY BYTE
3122 1C 02040 INC E ; AND POSITION SCREEN
3123 C9 02050 RET ; AND BACK TO CALLER
02060 ;
02070 ; #####
02080 ; HEXADECIMAL TO ASCII CONVERSION FOR TWO-BYTE WORDS
02090 ; #####
02100 ;
3124 F5 02110 HEXASC PUSH AF ; SAVE BYTE IN AF REG.
3125 E6F0 02120 AND 0FDH ; AND MASK OUT LOW NYBBLE
3127 0F 02130 RRCA ; BEGIN ROTATING BIT ...
3128 0F 02140 RRCA ; ... IN ORDER TO TEST...
3129 0F 02150 RRCA ; ... THE LOW NYBBLE ...
312A 0F 02160 RRCA ; ... ALL BY ITSELF.
312B C03931 02170 CALL HXAC ; BYTE RANGE EVALUATION
312E 12 02180 LD (DE),A ; DISPLAY ON THE SCREEN
312F 1C 02190 INC E ; AND MOVE TO NEXT POS'N
3130 F1 02200 POP AF ; RESTORE ORIGINAL BYTE
3131 E60F 02210 AND 0FH ; AND MASK OUT HIGH BITS
3133 C03931 02220 CALL HXAC ; RANGE EVALUATION ROUT.
3136 12 02230 LD (DE),A ; AND DISPLAY ON SCREEN
3137 1C 02240 INC E ; MOVE TO NEXT VIDEO POSN
3138 C9 02250 RET ; AND BACK TO CALLER
02260 ;
02270 ; #####
02280 ; HEX-ASCII RANGE EVALUATION FOR 0 TO 9 AND A TO F AREAS
02290 ; #####
02300 ;
02310 ;
3139 FEDA 02320 HXAC CP DAH ; CHECK AGAINST 10 DEC.
313B 3003 02330 JR NC,NEXTX ; IF >10 THEN ALPHA CHAR
313D C63D 02340 ADD A,30H ; ELSE NUMERIC - CONVERT
313F C9 02350 RET ; AND BACK TO CALLER
3140 C637 02360 NEXTX ADD A,37H ; ALPHABETIC - CONVERT
3142 C9 02370 RET ; AND BACK TO CALLER
02380 ;
02390 ; #####
02400 ; ASCII TO HEXADECIMAL CONVERSION FOR TWO-BYTE WORDS
02410 ; #####
02420 ;
02430 ;
3143 C06331 02440 ASCHEX CALL ACHX ; RANGE EVALUATION ROUT.
3146 4F 02450 LD C,A ; SAVE PART OF CONVERSION
3147 2B 02460 DEC HL ; AND GET NEXT BYTE
3148 C05B31 02470 CALL LLLLS ; EXECUTE LEFT ROTATES
314B 81 02480 AOD A,C ; AND ADD TO MAKE BYTE
314C 4F 02490 LD C,A ; SAVE IT BACK IN C REG.
314D 2B 02500 DEC HL ; AND MOVE TO NEXT BYTE
314E C06331 02510 AX2 CALL ACHX ; DO RANGE EVALUATION
3151 47 02520 LD B,A ; AND SAVE IT IN B REG.
3152 2B 02530 DEC HL ; GET FINAL BYTE READY
3153 C05B31 02540 CALL LLLLS ; AND EXECUTE LEFT ROTATE
3156 80 02550 AOD A,B ; CREATE COMPLETE BYTE
3157 47 02560 LD B,A ; AND SAVE IT IN B REG.
3158 C5 02570 PUSH BC ; GET 2 BYTES TO XFER
3159 E1 02580 POP HL ; TRANSFER TO HL REGISTER
315A C9 02590 RET ; AND BACK TO CALLER
315B C06331 02600 LLLLS CALL ACHX ; EVALUATE RANGE OF CHAR
315E 07 02610 RLCA ; BEGIN LEFT ROTATES ...
315F 07 02620 RLCA ; ... WHICH WILL POS'N...
3160 07 02630 RLCA ; ... BYTE READY FOR ...
3161 07 02640 RLCA ; ... CONVERSION.
3162 C9 02650 RET ; AND GO BACK TO CALLER
3163 7E 02660 ACHX LD A,(HL) ; GET BYTE FROM SCREEN
3164 FE40 02670 CP 40H ; CHECK AGAINST ALPHA
3166 3003 02680 JR NC,NEXTZ ; IF NUMERIC THEN JUMP
316B D63D 02690 SUB 30H ; ELSE NUMBER TO HEX
316A C9 02700 RET ; AND BACK TO CALLER
316B D637 02710 NEXTZ SUB 37H ; THEN IT'S ALPHA TO HEX
316D C9 02720 RET ; AND BACK TO CALLER
02730 ;
02740 ; #####
02750 ; ROUTINE TO ACCEPT DATA FOR INPUT TO DISPLAYED MEMORY
02760 ; LOCATIONS, WITH EXIT TO MENU UPDATE OR MEMORY INCREMENT
02770 ; #####
02780 ;
316E C0C331 02790 DATMOD CALL INPUT ; GET CHAR. FROM KEYBRD.
3171 FE2E 02800 CP '.' ; IS IT A PERIOD?
3173 C8B031 02810 JP Z,A0EXIT ; IF SO, EXIT TO ADDR MOD
3176 FE0D 02820 CP 00H ; IS IT A CARRIAGE RET?
317B 2B2F 02830 JR Z,NEXT0 ; IF SO, GO TO NEXT DATA
317A FE30 02840 CP 'G' ; BEGIN TESTING FOR RANGE
317C 3BFO 02850 JR C,DATMOD ; BACK IF <0 CHARACTER
317E FE67 02860 CP 67H ; CHECK AGAINST LC ALPHA
318D 30EC 02870 JR NC,DATMOD ; IF >F THEN GO BACK

```

Listing Continued . . .

It was natural that Radio Shack would choose an inexpensive storage medium, cassettes, to accompany their low cost microcomputer. Because ordinary cassette players are audio devices, the tape saving and loading routines were designed to be slow but sure. With care, the CTR series of recorders can be as reliable as any other storage system designed for the TRS-80.

The weakness of the tape process comes from the obvious mismatching of an audio device, of very limited precision, with a digital device of unyielding high-precision. Portable cassette recorders are intended to reproduce audio signals with a reasonable level of fidelity. What constitutes a reasonable level of fidelity is disputable, and only a person with a true tin ear would not be able to pick out a portable player, from amongst a group of high fidelity tape decks.

But even with this 'reasonable' fidelity, much of what we recognize as harmonies and instrumentation is perceptible only because we already have an acculturated comprehension of sound; and this directly influences what we believe we are hearing. Our minds, in conjunction with our ears - average, fill in, smooth over and forgive minuscule failings. We have internal mechanisms which remember our experiences.

The cassette load/save system consists of seven major elements:

### 1. Serialization.

The individual bytes of computer data are converted into a stream of individual bits. This is a completely digital process, and the timing is provided by the computer.

### 2. Audio Processing.

The signal is converted into a 'digital audio' wave for recording on tape decks of unknown polarity. In other words, a digital, one to zero, signal is converted to an audio, one to minus one to zero, signal. In this way, an 'upside down' signal looks the same to a computer as the original.

### 3. Recording.

The signal goes through the tape recorder's electronics, and is recorded on a thin strip of magnetic tape. The audio electronics round the wave's edge, and the limitations of the tape contribute noise to the signal.

## Continued Listing

```

3182 FE3A 02880 CP ' ; TEST FOR NUMERIC >9
3184 3B04 02890 JR C,INDATA ; IF <9 THEN GET DATA
3186 FE81 02900 CP 61H ; TEST FOR ALPHA <A
3188 3BE4 02910 JR C,DATMOO ; IF <A THEN GO BACK
318A 210A3F 02920 INDATA LD HL,3F06H ; READY THE SCREEN PTR.
3180 FS 02930 PUSH AF ; SAVE VALUE FROM KEYBRD
318E 7E 02940 LD A,(HL) ; GET LOWER ASCII NYBBLE
318F 32093F 02950 LD (3F09H),A ; AND PUT IN HIGH POS'N
3192 F1 02960 POP AF ; RESTORE VALUE FROM KBO
3193 FE60 02970 CP 60H ; CHECK AGAINST LC ALPHA
319S 3B02 02980 JR C,ZIPZY ; IF LC, THEN SKIP PAST
3197 0620 02990 SUB 20H ; ELSE CONVERT TO U.C.
3199 77 03000 ZIPZY LD (HL),A ; AND DISPLAY ON SCREEN
319A C04E31 03010 CALL AX2 ; SINGLE BYTE CONVERSION
3190 7C 03020 LD A,H ; SAVE ONE BYTE IN A REG.
319E FS 03030 PUSH AF ; SAVE THAT NOW ON STACK
319F 21063F 03040 LD HL,3F06H ; POINT HL TO SCREEN POSN
31A2 C04331 03050 CALL ASCHEX ; EVALUATE ADDRESS THERE
31AS F1 03060 POP AF ; RESTORE VALUE TO SHOW
31A6 77 03070 LD (HL),A ; AND PUT ON THE SCREEN
31A7 1B07 03080 JR EXIT1 ; NOW GO OUT AND STORE
31A9 21063F 03090 NEXTD LD HL,3F06H ; POINT HL TO SCREEN POSN
31AC C04331 03100 CALL ASCHEX ; CONVERT ADDRESS TO HEX
31AF 23 03110 INC HL ; MOVE OVER TO NEXT POSN
3180 11003F 03120 EXIT1 LD DE,3F00H ; POINT DE ADDR. DISPLAY
3183 C01731 03130 CALL WORDER ; AND CONVERT/DISPLAY IT
3186 1C 03140 INC E ; GO TO NEXT SCREEN POSN
3187 7E 03150 LD A,(HL) ; GET VALUE FROM MEMORY
3188 C02431 03160 CALL HEXASC ; AND CONVERT FOR DISPLAY
318B 1B81 03170 JR DATMOO ; AND BACK FOR SOME MORE
3180 21063F 03180 ADEXIT LD HL,3F06H ; POINT TO PRESENT ADDR.
31C0 C03B31 03190 JP NUM2 ; AND BACK TO UPDATE MENU
03200 ;
03210 ; #####
03220 ; COMPLETE KEYBOARD ROUTINE BELOW MAY BE CALLED BY NOT
03230 ; ONLY THE MONITOR ROUTINES, BUT ANY ROUTINES NEEDING AN
03240 ; UPPER/LOWER CASE, AUTOREPEAT, BEEP DRIVER. FOR FURTHER
03250 ; DESCRIPTION, SEE SUPPLEMENT TO CHAPTER ON KEYBOARD I/O
03260 ; #####
03270 ;
31C3 213640 03280 INPUT LO HL,4036H ; SAME AS LII BUFFER
31C6 01013B 03290 LO BC,3B01H ; FIRST KEYBOARD ROW
31C9 1600 03300 LD 0,0 ; COUNTER FOR COLUMNS
31CB 0A 03310 KEYPRS LD A,(BC) ; RETRIEVE ROW CONTENTS
31CC SF 03320 LD E,A ; SAVE IT TEMPORARILY
31CD A3 03330 AND E ; SET FLAGS FOR TESTING
31CE 201B 03340 JR NZ,STROKE ; NOT ZERO IF KEY PRESSED
31D0 77 03350 LD (HL),A ; SAVE CURRENT VALUE
31D1 14 03360 RECHK INC 0 ; INCREMENT ROW COUNTER
31D2 2C 03370 INC L ; INCREMENT STORAGE ADDR.
31D3 CB01 03380 RLC C ; GET NEXT KEYBOARD COL.
31D5 79 03390 LO A,C ; GET VALUE INTO ACCUM.
31D6 06B0 03400 SUB B0H ; LAST ROW IS 3B80 HEX
31D8 20F1 03410 JR NZ,KEYPRS ; NEXT CHECK IF NOT DONE
31DA 0607 03420 LD B,7 ; COUNTER OF KEYBRD ROWS
31DC 20 03430 CLRMEM DEC L ; START COUNTING BACKWARD
31DE B6 03440 ADD A,(HL) ; AND ADD IT UP IN ACCUM.
31E0 10FC 03450 DJNZ CLRMEM ; AND DO IT FOR 7 ROWS
31E1 A7 03460 AND A ; TEST FOR ANY KEY DOWN
31E3 3E00 03470 LD A,0 ; A=0, FLAGS ARE INTACT
31E4 321A40 03480 RET NZ ; BACK IF KEYS IN USE
31E7 C9 03490 LD (KPLACE),A ; ELSE DELAY GETS RESET
31E8 A6 03500 RET ; AND GO BACK ANYWAY
31E9 2B1B 03510 STROKE AND (HL) ; CHECK KEYSTROKE STORAGE
31EB 3A1A40 03520 JR Z,FOUND ; NEW KEY IF NOT SAME ONE
31EE 3C 03530 LD A,(KPLACE) ; NEW CHECK SPECIAL STORE
31EF 321A40 03540 INC A ; LET STORE = STORE + 1
31F0 321A40 03550 LD (KPLACE),A ; AND PUT IN BACK THERE
31F2 FEFF 03560 CP OFFH ; CHECK IF IT IS AT END
31F4 2B0B 03570 JR Z,DECA ; IF SO, THEN HOLD THERE
31F6 CS 03580 PUSH BC ; SAVE ROW COUNTER REG.
31F7 06FF 03590 LD B,OFFH ; GET DELAY VALUE INTO B
31F9 10FE 03600 TMRSTE DJNZ TMRSTE ; AND DELAY JUST A BIT
31FB C1 03610 POP BC ; RESTORE ROW COUNTER
31FC 1B03 03620 JR RECHK ; AND BACK TO CHECK NEXT
31FE 30 03630 DECA DEC A ; LET A = A - 1 (STORE)
31FF 321A40 03640 LD (KPLACE),A ; AND PUT IT IN STORAGE
3202 7B 03650 LD A,E ; GET KEYBOARD BYTE BACK
3203 73 03660 FOUND LD (HL),E ; STORE IT IS STROKE AREA
3204 7A 03670 LD A,D ; GET ROW COUNTER FROM D
3205 07 03680 RLCA ; AND BEGIN PROCESS OF...
3206 07 03690 RLCA ; ...CONVERTING IT...
3207 07 03700 RLCA ; ...TO AN OFFSET VALUE
3208 57 03710 LD 0,A ; AND PUT IT BACK IN D
3209 0E01 03720 LD C,1 ; GET NUMBER DNE READY
320B 79 03730 BACKUP LD A,C ; ACCUM. HAS C FOR MASK
320C A3 03740 AND E ; TEST IF C = KEYSTROKE
320D 2005 03750 JR NZ,AROUND ; IF NOT, THEN GO AROUND
320F 14 03760 INC 0 ; ELSE 0 = ROW + COLUMN
3210 CB01 03770 RLC C ; C SET TO NEXT COLUMN
3212 1B77 03780 JR BACKUP ; GO BACK AND TEST AGAIN
3214 3AB03B 03790 AROUND LD A,(3B80H) ; GET SHIFT ROW FDR TEST
3217 47 03800 LD B,A ; AND SAVE IT IN B REG.

```

## 4. Storage.

The tape sits on the shelf, affected by temperature and humidity, where its oxide coating may 'creep'. The tape may stretch or buckle or warp, and its upper and lower edges may become slightly feathered.

## 5. Playback.

The recorded signal, including warps, dropouts, feathering, creep and noise, is fed to the playback electronics. This audio circuit contributes further noise, providing a purely low grade audio signal to the computer.

## 6. Digital Processing.

The signal is received, its top and bottom edges are squared, and it is returned to the one to zero digital state. The timing is provided by the recorded tape in cooperation with timing loops provided by the computer.

## 7. De-Serialization.

A completed group of bits is assembled into an 8 bit byte for use by the CPU in determining synchronization, type of program, loading location, etc.

An oscilloscope representation of a digital signal being generated during the CSAVE routine is shown below. The signal changes from one to zero very crisply, spending only a few nanoseconds (not visible at all on the photo) in the transition area between one and zero. This true digital signal is measured just before the audio processing section sent to the cassette.



The audio processing is actually no more than a digital signal whose polarity is reversed. Two outputs of a latch are tied together, and as one goes high, the other is forced low. Between pulses, the signal floats to the middle. This is done by data bits 0 and 1 of Z59, which are alternately switched by the CPU during CSAVE. The digital signal as it leaves the cassette port is shown in the second photograph.

The sharp edges of the signal have been ever so slightly blurred, partly due to the capacitance

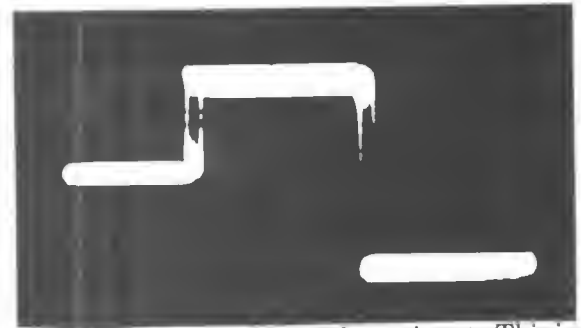
Listing Continued...

## Continued Listing

```

3218 7A      03910      LD      A,0          ; GET ROW COUNTER BACK
3218 C640    03920      ADD     A,40H        ; AND CONVERT TO ASCII
3218 FE60    03930      CP       60H         ; IS IT UP/LW/GRAFIX/CTRL
3210 3016    03940      JR       NC,JUMP1     ; GO OUT IF GRAPHICS
321F 57      03950      LD       0,A         ; SAVE PARTLY CONVERTED
3220 3A4038  03960      LD       A,(3840H)    ; GET VALUE FOUND 7TH ROW
3223 E610    03970      ANO      10H         ; CHECK IF DOWN ARROW
3225 2009    03980      JR       NZ,CTRL     ; IF SO, PRODUCE CONTROL
3227 7A      03990      LD       A,0         ; ELSE GET VALUE BACK
3228 C808    03900      RRC       9           ; 9 BUMPS INTO CARRY FLAG
322A 3830    03810      JR       C,GOAWAY     ; IF CARRY, THEN SHIFT
322C C620    03920      ADD     A,20H         ; IF NOT THEN LOWER CASE
322E 1839    03930      JR       GOAWAY     ; AND GET OUT OF ROUTINE
3230 7A-     03940      LD       A,0         ; IF CONTROL CODE, GET IT
3231 0640    03950      SUB      40H         ; GET RID OF ASCII MASK
3233 1834    03960      JR       GOAWAY     ; AND GET OUT OF ROUTINE
3235 0670    03970      JUMP1    SUB      70H     ; THE BALANCE OF THE
3237 3010    03980      JR       NC,JUMP2     ; ROUTINE BELOW UP TO
3239 C640    03990      ADD     A,40H         ; THE 9EEP SECTION IS
323B FE3C    04000      CP       3CH         ; VIRTUALLY IDENTICAL
3230 3802    04010      JR       C,JUMP3     ; TO THE KEYBOARD
323F EE10    04020      XOR      10H         ; DETERMINATION SUB-
3241 C909    04030      JUMP3    RRC       9           ; ROUTINE FOUND IN
3243 3024    04040      JR       NC,GOAWAY    ; RCH. THIS ENTIRE
3245 EE10    04050      XOR      10H         ; ROUTINE IS REPEATED
3247 1820    04060      JR       GOAWAY     ; BECAUSE IT CAN
3249 07      04070      JUMP2    RLCA      ; SERVE AS AN EXACT
324A C908    04080      RRC       9           ; REPLACEMENT FOR
324C 3000    04090      JR       NC,JUMP4     ; THE LEVEL II BASIC
324E 215832  04100      LD       HL,TABLET    ; ROUTINES WHEN (IF)
3251 4F      04110      LD       C,A         ; THE ROMS HAVE BEEN
3252 0600    04120      LD       9,0         ; REMOVED, REPLACED,
3254 09      04130      ADD     HL,9C         ; DISABLED, ETC.
3255 7E      04140      LD       A,(HL)       ; SEE KEYBOARD ROUTINE
3256 1811    04150      JR       GOAWAY     ; RUNDOWN ELSEWHERE
3258 3C-     04160      INC      A           ; IN THIS BOOK.....
3259 0000    04170      DEFW    0000H        ; CARR. RET. / CARR. RET.
325B 1F1F    04180      DEFW    1F1FH        ; CLEAR SCRIN / CLEAR 6CRN
325D 0101    04190      DEFW    0101H        ; BREAK KEY / BREAK KEY
325F 5818    04200      DEFW    185BH        ; EDIT ESCAPE / UP ARROW
3261 0A00    04210      DEFW    000AH        ; NO CHANGE / LINEFEED
3263 0818    04220      DEFW    1809H        ; BACKSP. LINE / BACKSP.
3265 0918    04230      DEFW    1909H        ; 32-CHAR MODE / HOR TAB
3267 2020    04240      DEFW    2020H        ; SPACE / SPACE
3269 57      04250      LD       0,A         ; SAVE VALUE IN 0 REG.
326A 018001  04260      LD       9C,190H     ; DEBOUNCE VALUE AND
326B 09      04270      DELAYS   OEC      9C   ; IMPLEMENTED AT THIS
326E 79      04280      LD       A,8         ; POINT FOR ACTUAL
326F 81      04290      OR       C           ; KEYBOARD SCAN WHEN
3270 20F8    04300      JR       NZ,DELAYS   ; CHARACTERS ARE
3272 7A      04310      LD       A,0         ; GET STORED VALUE BACK
3273 F5      04320      PUSH     AF          ; SAVE ACCUM. & FLAGS
3274 0640    04330      LD       9,40H       ; GET BEEP LENGTH VALUE
327B 3A3040  04340      LD       A,(403DH)    ; GET STATUS OF SCREEN
3278 E6F0    04350      ANO      0FDH        ; MASK SCREEN CHANGE
327B 67      04360      LD       H,A         ; STORE MS9 IN H REG.
327C F602    04370      OR       2           ; SET BIT 1 TO BE ON
327E 6F      04380      LD       L,A         ; STORE ALT. MS9 IN L REG
327F 70      04390      LD       A,L         ; GET ALT. MS9 TO OUTPUT
3280 03FF    04400      OUT      (OFFH),A     ; AND OUTPUT RISING WAVE
3282 7C      04410      LD       A,H         ; GET NORMAL MS9 CHAR.
3283 03FF    04420      OUT      (OFFH),A     ; AND OUTPUT FALLING WAVE
3285 CS      04430      PUSH     8C          ; SAVE NOTE LENGTH VALUE
3286 0640    04440      LD       8,40H       ; GET FREQUENCY DELAY
3288 10FE    04450      FREQCY   DJNZ    FREQCY ; AND WAIT A LITTLE WHILE
328A C1      04460      POP      8C          ; NOW RESTORE LENGTH VAL.
328B 10F2    04470      DJNZ     9EEPER      ; AND GO BACK THAT LENGTH
328D F1      04480      POP      AF          ; RESTORE ORIGINAL CHAR.
328E C9      04490      RET              ; AND BACK TO CALLER
328F 7E      04500      ;
3290 A7      04510      ; *****
3291 C8      04520      ; ROUTINE BELOW TAKES SIMPLE MESSAGES AND DISPLAYS THEM
3292 12      04530      ; *****
3293 23      04540      ;
3294 13      04550      DISPLY    LO      A,(HL) ; GET FIRST MSG CHARACTER
3295 18F8    04560      ANO      A           ; TEST IF 0 OR CHARACTER
3296 7E      04570      RET      Z           ; BACK TO CALLING ROUTINE
3297 3C      04580      LD       [0E],A      ; DISPLY THE CHARACTER
3298 0F      04590      INC      HL          ; GET NEXT MSG CHARACTER
3299 13      04600      INC      OE          ; GET NEXT DISPLY POS'N
329A 13      04610      JR       DISPLY      ; AND TEST NEXT CHARACTER
329B 18F8    04620      ;
329C 3C      04630      ; *****
329D 3C      04640      ; MESSAGES FOR DISPLY FOLLOW. IN A STRIPPED SOFTWARE
329E 3C      04650      ; MONITOR CONFIGURATION, THESE MESSAGES MAY BE ELIMINATED
329F 3C      04660      ; AND DISPLAY PARAMETERS CHANGED ELSEWHERE IN THE LISTING
329F 3C      04670      ; *****
329F 3C      04680      ;
329F 3C      04690      MSG01    DEFM    '<C>LOSTART, <D>I6K800T, <M>XONITOR, '
329F 3C      04700      DEFM    '<U>SERJUMP, <W>ARMSTART? '
329F 3C      04710      DEFM    8FH
329F 3C      04720      DEFM    00
329F 3C      04730      MSG02    DEFM    '290 REGISTERS AT MONITOR ENTRY: '

```



introduced by the cables and tape input. This is the first (and least significant) step in the extensive route of signal degradation.

```

32F6 20      04740      0EFM    '
3316 53      04750      0EFM    'SP=   IX=   IY=   AF=
3336 42      04760      0EFM    'BC=   OE=   HL=   RI=
3356 41      04770      0EFM    'ALTERNATE REGISTERS AT MONITOR E'
3376 4E      04780      0EFM    'NTRY (NOT USED IN LEV II BASIC):'
3396 41      04790      0EFM    'AF=   BC=   OE=   HL=
33B6 20      04800      0EFM    '
33D6 55      04810      0EFM    'USE PERIOD (.) TO ENTER AN ADDRE'
33F6 53      04820      0EFM    'SS, SLASH (/) TO ENTER DATA. TO'
3416 45      04830      0EFM    'EXIT TO MONITOR MENU, TYPE STAR'
3436 28      04840      0EFM    '(*), TO EXECUTE AT ADDRESS SHOWN'
3456 4F      04850      0EFM    'ON SCREEN, TYPE EXCLAMATION POIN'
3476 54      04860      0EFM    'T (!)
3496 00      04870      0EF8    00
3497         04880      EQU     $
3497         04890      ;
3497         04900      ENO
0000         02660      02440 02510 02600
A00M00 3007 01620      01700 01720 01760
A0EXIT 3190 03190      02810
AROUND 3214 03790      03750
ASCHEX 3143 02440      01860 01890 03050 03100
AX2     314E 02510      03010
BACKUP 3209 03730      03790
BEEPER 327F 04390      04470
CHECK   30CC 01520      01970
CLEAR   303E 00640      00710
CLRMEM  310C 03430      03450
CLS      300C 00310      01680
CTRL    3230 03940      03980
COLD     303C 00620      00460
GATM00  316E 02790      01660 02850 02870 02910 03170
OECA     31FE 03630      03570
DELAYS   3260 04270      04300
DISPLY   32BF 04550      00430 01100 04610
EXEC     3110 01880      01640
EXIT1    3180 03120      03080
FOUND    3203 03660      03520
FREQCY   3288 04450      04450
GOAWAY   3269 04250      03910 03930 03960 04040 04060 04150
HEXASC   3124 02110      01560 02010 02030 03160
HXAC     3139 02320      02170 02220
INDATA   318A 02920      02890
INPUT    31C3 03260      00440 01620 02790
JUMP1    3235 03970      03940
JUMP2    3249 04070      03980
JUMP3    3241 04030      04010
JUMP4    324E 04100      04090
KEYPRS   31C9 03310      03410
KPLACE   401A 00160      03490 03530 03550 03640
LLLLS    3159 02600      02470 02540
LODP1    3051 00730      00750
MENU     3022 00440      00550
MONITOR  306F 01090      00500
MSG01    3297 04690      00410
MSG02    3206 04730      01080
NEXT0    31A9 03090      02830
NEXTX    3140 02360      02330
NEXTZ    316B 02710      02680
NUM2     3109 01960      03190
NUMBER   30FB 01770      01740
RECHCK   3101 03360      03620
STROKE   31EB 03510      03340
TABLET   3259 04170      04100
TMWSTE   31F9 03600      03600
USER     3058 00830      00520
WARM     3064 00950      00540
WORDER   3117 01970      01150 01190 01210 01230 01250 01270 01290
01340 01400 01470 01490 01510 01530 03130
XXXXXX  3497 04880
ZIP8Y    310A 01850      01830
ZIPZY    3199 03000      02980

```

The worst signal abuse takes place during the taping process. The reasonably digital signal is recorded with the poor electronics of a portable tape recorder, and the sharp-edged waves are rounded off by the natural limitations of the tape itself; examine Photo 6-4. Also visible in the photo are residual noise (tape hiss) and the high-frequency recording bias signal.

There is also an unexpected interreaction between the computer's output wiring and some tape recorders that produces a low-pitched hum, called a ground loop. The good data signal can ride on this ground-loop hum to result in sensitive volume settings — too high or low a volume during playback will cause the top or bottom of the digital waveform to travel out of the range of the digital processing.

This digital processing redeems quite a bit of the audio signal, turning it back into a usable digital waveform much of the time. Photo 6-5 shows the results of the reshaping process; it's a fairly good signal that the CPU finally receives and must interpret as data.

Now the names and descriptions of the seven CLOAD culprits:

1. Head misalignment. This is the main cause of bad loads, because misalignment severely cuts the essential high frequencies. The CTR-80 already has a provision for adjusting the playback head. If you have a CTR-41 (or other recorder), drill a hole directly over the playback head adjustment screw (under 1. Head misalignment. This is the main cause of bad loads, because misalignment severely cuts the essential high frequencies. The CTR-80 already has a provision for adjusting the playback head. If you have a CTR-41 (or other recorder), drill a hole directly over the playback head adjustment screw (under the letters ERY in 'battery' on the CTR-41 face plate), and adjust with a small Phillips screwdriver.
2. Speed variations. This one is not obvious, but note that a five percent variation in recorder speed can cause a bad load, especially with long BASIC lines during CLOAD.
3. Bad tape. There is no reason to use low-grade tape, just as there is no reason to buy the best audio tape. Get a good commercial grade, and standardize with it.
4. Dirty head. Clarity and volume are cut down when the head is dirty. Clean it and all parts that contact the tape with isopropyl alcohol.

5. Starting too soon. The beginning of most tape — especially leaderless tape — is often slightly crumpled, and data can be lost right at the start. Count off ten seconds.

6. Magnetized head. Those who depend on cassettes will use the machine often, and the head will build up residual magnetism. Obtain a cassette demagnetizer (degausser) and use it often.

7. Software. Early Level II BASIC ROMs have problems because the timing loops were not written ideally for low-grade audio use. These can be upgraded with new ROMs or the Radio Shack XRX cassette modification. Note, however, that removal of the XRX modification is necessary for use with high/low speed hardware modifications (except the Archbold 1981-82 kit).

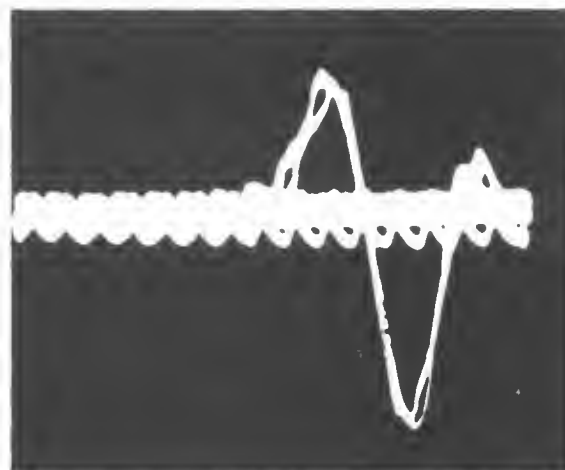


Photo 6-4. Rounded-off digital signal with tape hiss.

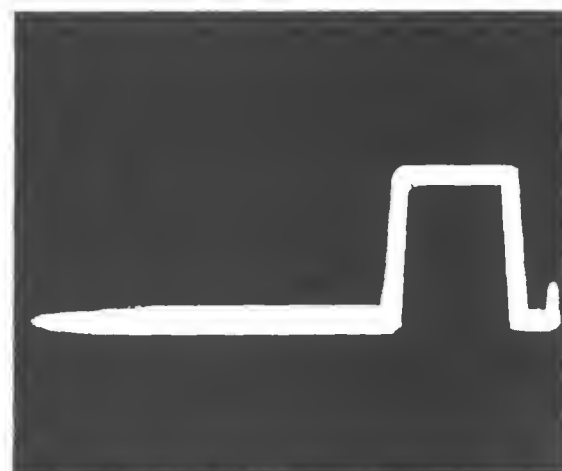


Photo 6-5. Reshaped wave that the CPU finally sees.

# NOTES



---

## Controlling the World

The world is out there, waiting to be controlled by your TRS-80. Really. Most of the partly-mechanized daily routine that you find yourself doing by hand might be connected to the '80. Not that it should be, mind you, but it could be. You could successfully operate a clothes washer and drier, a dishwasher, an automatic furnace, an air conditioner, or many other electromechanical devices; turn lights, televisions, and radios on and off, as well as replace your ordinary wake-up alarm; dial the phone, or answer it in a synthetic voice; even act as a digital burglar alarm. But would you really want to?

Admittedly, microprocessors are making their way into more and more appliances, including such diverse items as automobile engines and electric razors. But remember that the TRS-80 is a spectacular *general-purpose* device; although it might do all sorts of control work, it's best suited to applications calling for a large library of different electronic tasks, most with crucial human interaction. Aside from machine-oriented tasks like word processing, calculating, and printing, then, these applications include (generally speaking) measurement, signalling, and cloning.

Measurement involves the evaluation of a real-world occurrence in real time: checking temperature, counting events, comparing relative amounts of light or voltage or sound. Some of these measurements, like counting

events, are inherently digital – an event (say, a person walking through a door) either is taking place or it isn't. A door is closed or it's open. And so on. Other measurements are relative or quantitative, consisting of small increments or continuous change, such as the pressure, the rate of flow, or the quantity of water in a pipe.

Signalling is somewhat the opposite of measurement: a user is informed when some activity has been completed, or when a particular condition in the environment has been reached – including such things as the completion of a mailing list sort or someone breaking and entering.

Cloning is a rather odd phrase, but it's what I would like to call the computer's ability to create precise duplicates of some target object or activity. If that sounds a bit too philosophical, then think of it this way: the computer is fast, which means if it is given a task, it can complete it quickly. The computer is capable of calculating the parameters of its task with enormous accuracy. And finally, the computer works in minute, definable, and identical increments. Simply stated, we can command a computer to do work which, barring glitches and bugs, will be identical every time.

This last concept is the reason microprocessors have become the favored design tools for machining, measurement, and even the creation of music. Jigs wear out, so that tolerances change; but computer programs can be self-correcting. Measuring devices can go out

of alignment: again, computers can be programmed to cross-check and correct these errors. And finally, where electronic design has been sloppy as a result of inaccuracies in the electronic parts themselves (such as in synthesized music), computers can provide the advantages of precise replication where it was not possible before.

The general approach to discrete, digital events is discussed in Chapter (?), where input/output ports are presented. The hardware of interfacing digital inputs and outputs to electrical appliances and other 'real-world' devices is brilliantly described in *TRS-80 Interfacing* (Volumes 1 and 2 — see Bibliography; especially check Volume 2, Chapter 1).

This Chapter will present a few real-world interfacing projects, but will touch only lightly on the theory involved. If you plan to put together your own interfaces, turn to the references cited. Beyond that, here are some rules of thumb before you begin considering interfacing schemes or parts purchases:

### Output Interfacing

TTL-level integrated circuits of the type used in your TRS-80 are capable of driving (running) little more than other TTL circuits like themselves or an occasional light-emitting diode (LED). Don't hook your computer expansion card onto any home-made electronic board unless it has TTL inputs; consider the computer's bus to be its most delicate hardware.

For interfacing purposes, integrated circuit peripheral drivers are great. Use them to light small bulbs, turn on miniature relays, and to operate other low-voltage applications needing only limited current. Type 75452 is reliable and cheap, and takes abuse; it's the circuit used by the TRS-80 to run the video dot output and cassette relay. For LEDs, use individual transistors or the inexpensive type 500, 501 and 502 digit and segment drivers.

Isolation of hazardous voltages is essential for external equipment running from your computer. Consider anything plugged into house current to be potentially hazardous, because there's nothing like a power surge or unexpected short circuit to fry your '80 and maybe you too. Use opto-isolators and — or high-current relays for running electric stoves or even AC light bulbs.

Always read the specifications of both the equipment you plan to interface and the device

that's going to do the interfacing. Look for:

- Average voltage and maximum voltage of the interfacing device, and operating voltage of the equipment to be run; the figure is given in volts (V), direct current volts (VDC), or alternating current volts (VAC). The interfacing device must always have a rating higher than the equipment to be run.
- Average current and surge current of the the interfacing device, and operating current of the equipment to be run; the figure is given in amps (A) or milliamps (mA). The interfacing device must always have a rating higher than the equipment to be run.
- Isolation voltage of the interfacing device. This device must have a rating roughly 67 percent higher than the equipment to be run.

Actually, interfacing peripheral equipment is one of the easiest things you can do with your TRS-80. Below is a simple schematic; it shows one latched output line from the computer, and how it might drive:

- (a) an LED
- (b) a relay
- (c) an ordinary house lamp
- (d) a motor
- (e) a high-voltage circuit.

You wouldn't want to use this single line to do all these things, but you might want to combine the LED, house lamp, and motor. That way you could have an indicator near the computer that the motor is on, a bright lamp outside a building to indicate the motor is on, and the motor itself would go on.

The point of this schematic is that it can be tapped at any point on the line of parts shown; use the parts only as far as you need them:

### Input Interfacing

Counting events or determining occurrences has an easier group of rules: TTL level signals can interface right to other TTL devices with no work; just don't connect outputs together.

Other on-off signals are interfaced with a device or group of devices which can shape the incoming signal into a neat, tight square wave, at TTL level. Such a device is called a Schmitt trigger, and is available as type 7414 for a few cents.

If the signal is in the range of 4 to 7 volts, the Schmitt trigger will transform it to a fast-moving digital signal. If the signal occurs very often or is erratic or unstable in its rising voltage, then

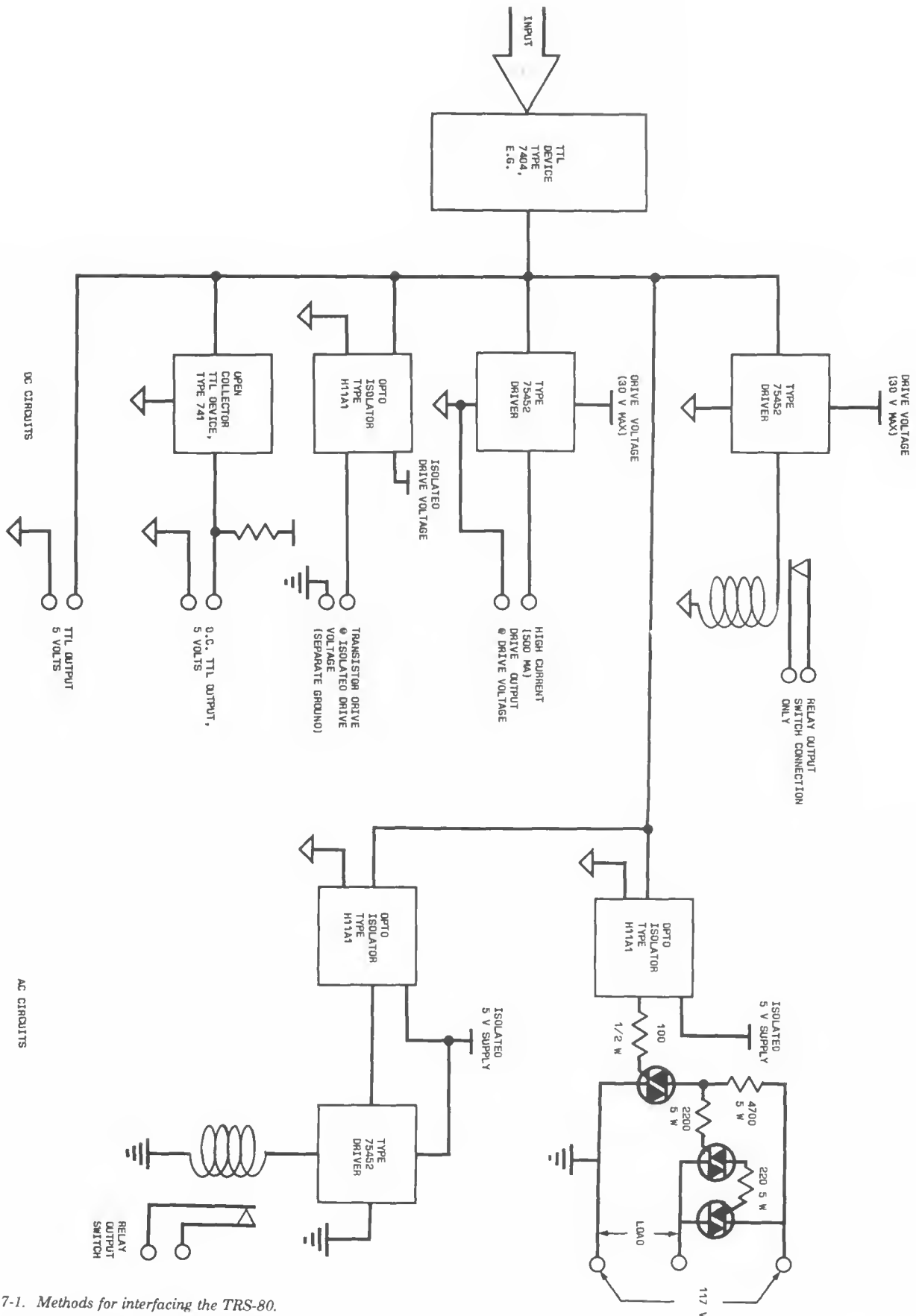


Figure 7-1. Methods for interfacing the TRS-80.

## D-to-A and A-to-D Conversion

additional hardware or very tolerant software must be used. Use optical isolators to feed the computer or relays for interfacing higher voltages.

For AC input, the signal may be transformed to about 3 to 6 volts, rectified to DC and filtered only if the signal is slow-moving (once a minute or so). Then the signal can be fed to a Schmitt trigger to the computer interface. Otherwise, the AC information can only be transformed to a lower voltage and its pulses (60 per second if it is ordinary house current) counted either by hardware or software. Don't interface AC if you can avoid it, because it can be a genuine pain in the bytes. Instead, have the AC run a fast relay and identify when the relay turns on and off.



### A to D, D to A

#### DAC0808, DAC0807, DAC0806 8-bit D/A converter

##### general description

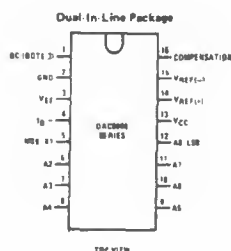
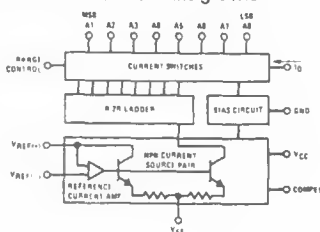
The DAC0808 series is an 8-bit monolithic digital-to-analog converter (DAC) featuring a full scale output current settling time of 150 ns while dissipating only 33 mW with  $\pm 5V$  supplies. No reference current ( $I_{REF}$ ) trimming is required for most applications since the full scale output current is typically  $\pm 1$  LSB of 255  $I_{REF}/256$ . Relative accuracies of better than  $\pm 0.19\%$  assure 8-bit monotonicity and linearity while zero level output current of less than  $4 \mu A$  provides 8-bit zero accuracy for  $I_{REF} \geq 2$  mA. The power supply currents of the DAC0808 series are independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.

The DAC0808 will interface directly with popular TTL, DTL or CMOS logic levels, and is a direct replacement for the MC1508/MC1408. For higher speed applications see DAC0800 data sheet.

##### features

- Relative accuracy:  $\pm 0.19\%$  error maximum (DAC0808)
- Full scale current match:  $\pm 1$  LSB typ
- 7 and 6-bit accuracy available (DAC0807, DAC0806)
- Fast settling time: 150 ns typ
- Noninverting digital inputs are TTL and CMOS compatible
- High speed multiplying input slew rate: 8 mA/ $\mu s$
- Power supply voltage range:  $\pm 4.5V$  to  $\pm 18V$
- Low power consumption: 33 mW @  $\pm 5V$

##### block and connection diagrams



##### typical application

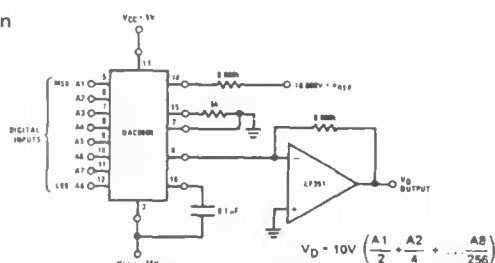


FIGURE 1.  $\pm 10V$  Output Digital to Analog Converter

##### ordering information

ACCURACY	OPERATING TEMPERATURE RANGE	ORDER NUMBERS*			
		D PACKAGE (D16C)	J PACKAGE (J16A)	N PACKAGE (N16A)	
8 bit	$-55^{\circ}C \leq T_A \leq +125^{\circ}C$	DAC0808LD	LM1508D-8	DAC0808LJ	LM1508J-8
8 bit	$0^{\circ}C \leq T_A \leq +75^{\circ}C$			DAC0808LCJ	LM1408J-8
7 bit	$0^{\circ}C \leq T_A \leq +75^{\circ}C$			DAC0807LCJ	LM1408J-7
6 bit	$0^{\circ}C \leq T_A \leq +75^{\circ}C$			DAC0806LCJ	LM1408J-6

\*Note: Devices may be ordered by using either order number.

Figure 7-2. National Semiconductor data sheet example.

Signals from photocells and similar resistive devices can be fed through Wein bridges or merely fed into an operational amplifier. Again, the references such as *TRS-80 Interfacing* and *Engineer's Notebook* contain plenty of details on interfacing on-off signals.

## D-to-A and A-to-D Conversion

Although some human events occur in on-off groups, most of life is pliable, elusive, and relative. It works by image and analogy, not by counting. It is an analog world, and the computer is a digital device. Faced with this dilemma, two important groups of electronic circuits have been developed: the digital-to-analog converter (D/A converter) and the analog-to-digital converter (A/D converter).

The first class accepts a parallel digital input a given number of bits wide, and converts that to individual steps. For  $N$  steps, the number of distinct voltages or currents is 2 to the power  $N$ . The greater the number of bits, the lesser the relative size of the steps, to the point where the distinction between steps becomes insignificant.

For an ordinary 8-bit data bus such as that on the TRS-80, the available voltage can be divided into 256 parts; for five-volt circuitry, this is 0.0195 volts per step, starting at zero. If greater accuracy is essential, then 12-bit converters can be used (at greater expense), fed by one and one-half bytes of data from the computer. This provides 4,096 steps at 0.0012 volts per step. The most common 8-bit type is the DAC0808 (National Semiconductor) or the MC1408L8 (Motorola); data sheets are found in an Appendix.

With present technology, conversion from analog signals to digital ones is much more

##### application hints

###### REFERENCE AMPLIFIER DRIVE AND COMPENSATION

The reference amplifier provides a voltage at pin 14 for converting the reference voltage to a current, and a turn-around circuit or current mirror for feeding the ladder. The reference amplifier input current,  $I_{14}$ , must always flow into pin 14, regardless of the set-up method or reference voltage polarity.

Connections for a positive voltage are shown in Figure 7. The reference voltage source supplies the full current  $I_{14}$ . For bipolar reference signals, as in the multiplying mode,  $R_{15}$  can be tied to a negative voltage corresponding to the minimum input level. It is possible to eliminate  $R_{15}$  with only a small sacrifice in accuracy and temperature drift.

The compensation capacitor value must be increased with increases in  $R_{14}$  to maintain proper phase margin; for  $R_{14}$  values of 1, 2.5 and 5 k $\Omega$ , minimum capacitor values are 15, 37 and 75 pF. The capacitor may be tied to either  $V_{EE}$  or ground, but using  $V_{EE}$  increases negative supply rejection.

**absolute maximum ratings** ( $T_A = 25^\circ\text{C}$  unless otherwise noted)

Power Supply Voltage		Power Dissipation (Package Limitation)	
VCC	5.5 VDC	Cavity Package	1000 mW
VEE	-16.5 VDC	Derate above $T_A = 25^\circ\text{C}$	67 mW/ $^\circ\text{C}$
Digital Input Voltage, V5-V12	-10 VDC to +18 VDC	Operating Temperature Range	
Applied Output Voltage, V <sub>O</sub>	-11 VDC to +18 VDC	DAC0808L	$-55^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$
Reference Current, I <sub>14</sub>	5 mA	DAC0808LC Series	$0 \leq T_A \leq +75^\circ\text{C}$
Reference Amplifier Inputs, V14-V15	VCC, VEE	Storage Temperature Range	$-65^\circ\text{C}$ to $+150^\circ\text{C}$

**electrical characteristics**

(VCC = 5V, VEE = -15 VDC, VREF/R14 = 2 mA, DAC0808L:  $T_A = -55^\circ\text{C}$  to  $+125^\circ\text{C}$ , DAC0808LC, DAC0807LC, DAC0806LC,  $T_A = 0^\circ\text{C}$  to  $+75^\circ\text{C}$ , and all digital inputs at high logic level unless otherwise noted.)

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
$E_r$ Relative Accuracy (Error Relative to Full Scale I <sub>D</sub> )	(Figure 4)				%
DAC0808L (LM1508 8),				$\pm 0.19$	%
DAC0808LC (LM1408 8)				$\pm 0.39$	%
DAC0807LC (LM1408 7), (Note 1)				$\pm 0.78$	%
DAC0806LC (LM1408 6), (Note 1)					%
Settling Time to Within 1/2 LS8 (Includes t <sub>PLH</sub> )	$T_A = 25^\circ\text{C}$ (Note 2), (Figure 5)		150		ns
t <sub>PLH</sub> , t <sub>PHL</sub> Propagation Delay Time	$T_A = 25^\circ\text{C}$ , (Figure 5)		30	100	ns
TCIO Output Full Scale Current Drift			$\pm 20$		ppm/ $^\circ\text{C}$
MSB Digital Input Logic Levels	(Figure 3)				
V <sub>IH</sub> High Level, Logic "1"		2			V <sub>OC</sub>
V <sub>IL</sub> Low Level, Logic "0"				0.8	V <sub>DC</sub>
MSB Digital Input Current	(Figure 3)				
High Level	V <sub>IH</sub> = 5V		0	0.040	mA
Low Level	V <sub>IL</sub> = 0.8V		-0.003	-0.8	mA
I <sub>15</sub> Reference Input Bias Current	(Figure 3)		-1	-5	$\mu\text{A}$
Output Current Range	(Figure 3)				
	VEE = -5V	0	2.0	2.1	mA
	VEE = -15V, $T_A = 25^\circ\text{C}$	0	2.0	4.2	mA
I <sub>D</sub> Output Current	VREF = 2.000V R14 = 1000 $\Omega$ , (Figure 3)	1.9	1.99	2.1	mA
Output Current, All Bits Low	(Figure 3)		0	4	$\mu\text{A}$
Output Voltage Compliance	$E_r \leq 0.19\%$ , $T_A = 25^\circ\text{C}$				
Pin 1 Grounded, VEE Below -10V				-0.55, +0.4 -5.0, +0.4	V <sub>DC</sub> V <sub>OC</sub>
SR <sub>REF</sub> Reference Current Slew Rate	(Figure 6)		8		mA/ $\mu\text{s}$
Output Current Power Supply Sensitivity	-5V $\leq$ VEE $\leq$ -16.5V		0.05	2.7	$\mu\text{A/V}$
Power Supply Current (All Bits Low)	(Figure 3)				
I <sub>CC</sub>			2.3	22	mA
I <sub>EE</sub>			-4.3	-13	mA
VCC Power Supply Voltage Range	$T_A = 25^\circ\text{C}$ , (Figure 3)	4.5	5.0	5.5	V <sub>OC</sub>
VEE		-4.5	-15	-16.5	V <sub>DC</sub>
Power Dissipation					
All Bits Low	VCC = 5V, VEE = -5V		33	170	mW
	VCC = 5V, VEE = -15V		106	305	mW
All Bits High	VCC = 15V, VEE = -5V		90		mW
	VCC = 15V, VEE = -15V		160		mW

Note 1: All current switches are tested to guarantee at least 50% of rated current.

Note 2: All bits switched.

Note 3: Range control is not required.

**application hints** (Continued)

A negative reference voltage may be used if R14 is grounded and the reference voltage is applied to R15 as shown in Figure 8. A high input impedance is the main advantage of this method. Compensation involves a capacitor to VEE on pin 16, using the values of the previous paragraph. The negative reference voltage must be at least 4V above the VEE supply. Bipolar input signals may be handled by connecting R14 to a positive reference voltage equal to the peak positive input level at pin 15.

When a DC reference voltage is used, capacitive bypass to ground is recommended. The 5V logic supply is not recommended as a reference voltage. If a well-regulated 5V supply which drives logic is to be used as the reference, R14 should be decoupled by connecting it to 5V through another resistor and bypassing the junction of the 2 resistors with 0.1  $\mu\text{F}$  to ground. For reference

voltages greater than 5V, a clamp diode is recommended between pin 14 and ground.

If pin 14 is driven by a high impedance such as a transistor current source, none of the above compensation methods apply and the amplifier must be heavily compensated, decreasing the overall bandwidth.

**OUTPUT VOLTAGE RANGE**

The voltage on pin 4 is restricted to a range of -0.6 to 0.5V when VEE = -5V due to the current switching methods employed in the DAC0808.

The negative output voltage compliance of the DAC0808 is extended to -5V where the negative supply voltage is more negative than -10V. Using a full-scale current of 1.992 mA and load resistor of 2.5 k $\Omega$  between pin 4 and ground will yield a voltage output of 256 levels between 0 and -4.980V. Floating pin 1 does not affect

the converter speed or power dissipation. However, the value of the load resistor determines the switching time due to increased voltage swing. Values of R<sub>L</sub> up to 500 $\Omega$  do not significantly affect performance, but a 2.5 k $\Omega$  load increases worst-case settling time to 1.2  $\mu\text{s}$  (when all bits are switched DN). Refer to the subsequent text section on Settling Time for more details on output loading.

**OUTPUT CURRENT RANGE**

The output current maximum rating of 4.2 mA may be used only for negative supply voltages more negative than -7V due to the increased voltage drop across the resistors in the reference current amplifier.

**ACCURACY**

Absolute accuracy is the measure of each output current level with respect to its intended value, and is dependent upon relative accuracy and full-scale current drift. Relative accuracy is the measure of each output current level as a fraction of the full-scale current. The relative accuracy of the DAC0808 is essentially constant with temperature due to the excellent temperature tracking of the monolithic resistor ladder. The reference current may drift with temperature, causing a change in the absolute accuracy of output current. However, the DAC0808 has a very low full-scale current drift with temperature.

The DAC0808 series is guaranteed accurate to within  $\pm 1.2$  LS8 at a full-scale output current of 1.992 mA. This corresponds to a reference amplifier output current drive to the ladder network of 2 mA, with the loss of 1 LS8 (8  $\mu\text{A}$ ) which is the ladder remainder shunted to ground. The input current to pin 14 has a guaranteed value of between 1.9 and 2.1 mA, allowing some mismatch in the NPN current source pair. The accuracy test circuit is shown in Figure 4. The 12-bit converter is calibrated for a full-scale output current of 1.992 mA. This is an optional step since the DAC0808 accuracy is essentially the same between 1.5 and 2.5 mA. Then the DAC0808 circuits' full-scale current is trimmed to the same value with R14 so that a zero value appears at the error amplifier output. The counter is activated and the error band may be displayed on an oscilloscope, detected by comparators, or stored in a peak detector.

Two 8-bit D-to-A converters may not be used to construct a 16-bit accuracy D-to-A converter. 16-bit accuracy implies a total error of  $\pm 1/2$  of one part in 65,536, or  $\pm 0.00076\%$ , which is much more accurate than the  $\pm 0.019\%$  specification provided by the DAC0808.

**MULTIPLYING ACCURACY**

The DAC0808 may be used in the multiplying mode with 8-bit accuracy when the reference current is varied over a range of 256:1. If the reference current in the multiplying mode ranges from 16  $\mu\text{A}$  to 4 mA, the additional error contributions are less than 1.6  $\mu\text{A}$ . This is well within 8-bit accuracy when referred to full-scale.

A monotonic converter is one which supplies an increase in current for each increment in the binary word. Typically, the DAC0808 is monotonic for all values of reference current above 0.5 mA. The recommended range for operation with a DC reference current is 0.5 to 4 mA.

**SETTLING TIME**

The worst-case switching condition occurs when all bits are switched DN, which corresponds to a low-to-high transition for all bits. This time is typically 150 ns for settling to within  $\pm 1/2$  LS8, for 8-bit accuracy, and 100 ns to  $1/2$  LS8 for 7 and 6-bit accuracy. The turn OFF is typically under 100 ns. These times apply when R<sub>L</sub>  $\leq$  500 $\Omega$  and C<sub>O</sub>  $\leq$  25 pF.

Extra care must be taken in board layout since this is usually the dominant factor in satisfactory test results when measuring settling time. Short leads, 100  $\mu\text{F}$  supply bypassing for low frequencies, and minimum scope lead length are all mandatory.

complicated. Most commonly used is a technique which compares the input voltage to known voltages inside the converter. When a match is found, a binary version of the input condition is fed to the computer bus. This process is slow when compared to computer speeds, and such devices usually have start-of-conversion and end-of-conversion signals so the computer will not receive false data by attempting to read the converter's information during the conversion process. The commonly used 8-bit type is ADC0800 (National Semiconductor); see the Appendix.

### Music and Sound Effects

There are other ways of producing sound using the TRS-80 which do not involve exclusively software. The software-only approach may be capable of producing sound effects, but in the TRS-80 configuration it cannot provide any

spatial effects (it is monaural only), nor can it offer a large variety of textures. For these, the TRS-80 user must turn to a little extra hardware.

Simplest among the extra hardware is a latched output address. The information written to the address appears at the output of the latch. Write to the address fast enough, and sound is produced, because the latch acts as a kind of electronic window to that single memory location. Feed that digital activity through a few resistors to blend the sounds, and run that to a stereo amplifier. Voila! Spatial sound.

The circuit presented here contains an 8-bit latch (Z1), and address decoder (Z3, Z4 and Z4, mapped to 4FFF), and three-state output buffer (Z2). The computer data is latched into Z1 when WRite to 4FFF appears at Z3/4/5. Depending on the data at Z1, any or all of the four buffers in Z2 may be turned on or off.

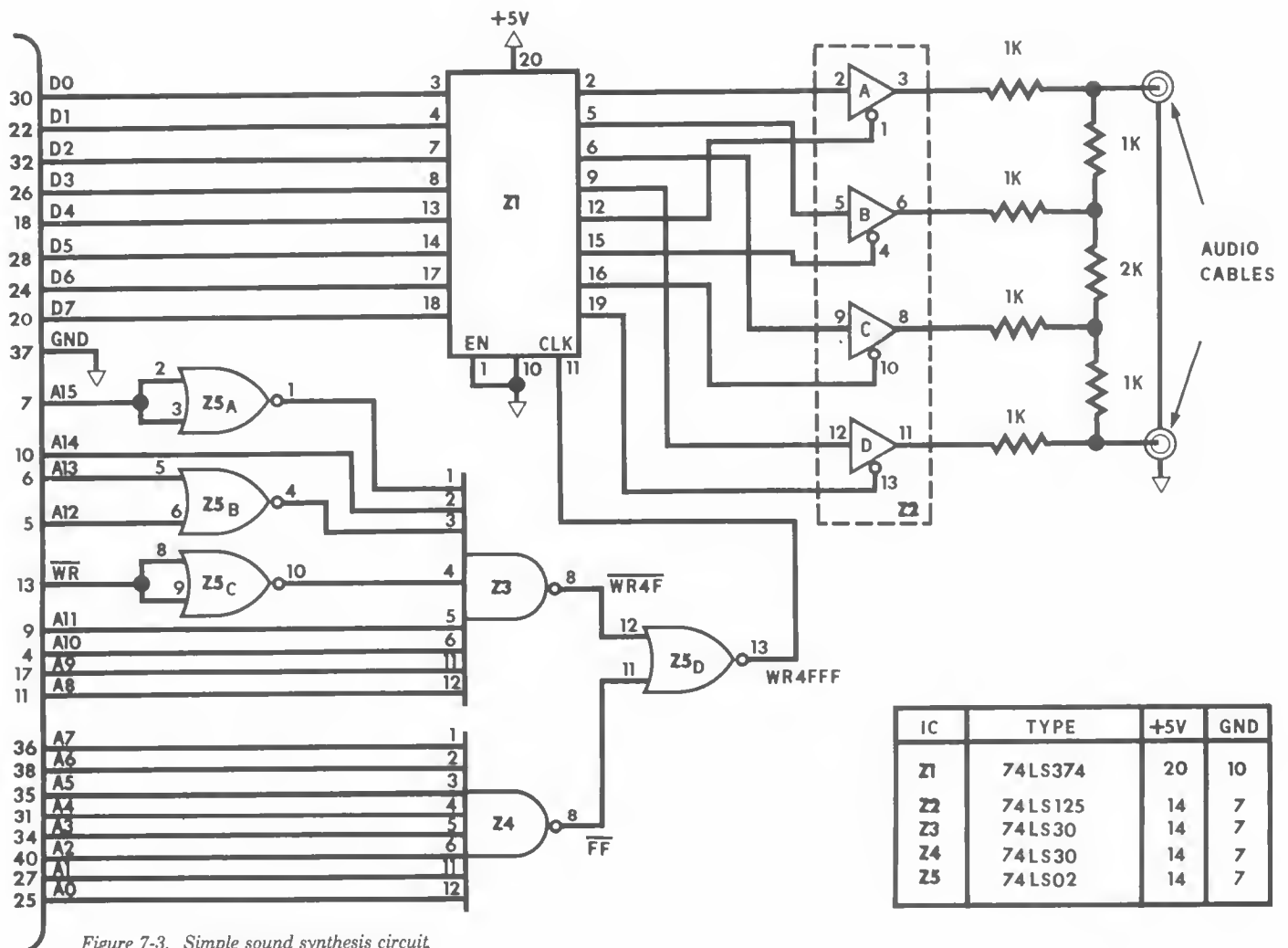


Figure 7-3. Simple sound synthesis circuit.

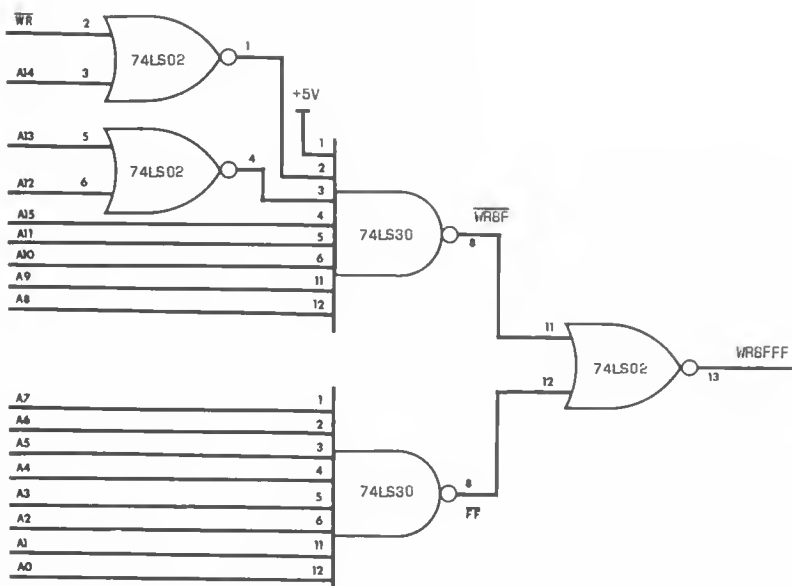


Figure 7-4. Alternate decoding scheme.

An alternate decoding (8FFF) is provided, and is recommended for some of the musical selections presented. The selections include my own arrangement of 'God Rest Ye Merry Gentlemen', first presented in the Christmas 1980 issue of *80 Microcomputing*; and 'Your California', a suite by David Gunn, originally composed for viola duo. The latter group is written for use with the *Exatron Stringy-Floppy*, and will continuously repeat the four parts of the suite.

There are two versions of the software; both are identical, except that the second is a two-voice program intended for pieces such as 'Your California'. Because the TRS-80 is slow (1.77 MHz) and the Z-80 is slow (an enormous number of machine cycles per instruction when compared with other microprocessor families), four-part music will reproduce in the low register. The two-part music is much more satisfying musically.



Figure 7-5. Music notation for people.

The image displays a musical score for a computer, consisting of multiple staves. The notation includes various musical symbols such as notes, rests, and slurs. There are several instances of triplets marked with a '3' and a bracket. The score is written in a standard musical notation style, with treble and bass clefs. The music appears to be a single melodic line, possibly for a computer-generated sound effect or a simple melody. The notation is clear and legible, with a focus on the rhythmic and melodic structure of the piece.

Figure 7-6. The same music for the computer.

```

10 REM * CLEARING SPACE FOR PROGRAM CHAINING SEQUENCE
20 POKE16634,PEEK(16634)+16 : REM * SIMPLE VARIABLE POINTER
30 CLEAR50 : REM * CLEAR STRING SPACE AND RESET POINTERS
40 @LOAD3 : REM * BRING NEXT PROGRAM IN FROM STRINGY FLOPPY

```

```

10 OUT254,2 : CLS : PRINT"READING.." : REM * OUT254,2 = HISPEED
20 X = 24578 : REM * THIS IS MEM. POSITION OF VOICE NUMBER ONE
30 READ A : IF A = 999 THEN 50 ELSE POKE X,A : REM * EXIT @ 999
40 X = X + 4 : GOTO 30 : REM * VOICE APPEARS EVERY 4 MEM LOC'NS
50 X = 24579 : REM * THIS IS MEM. POSITION OF VOICE NUMBER TWO
60 READ A : IF A = 999 THEN 80 ELSE POKE X,A : REM * PITCH POKE
70 X = X + 4 : GOTO 60 : REM * VOICE APPEARS EVERY 4 MEM LOC'NS
80 X = 24576 : REM * THIS IS THE BEGINNING VALUE FOR DURATIONS
90 READ A : IF A = 999 THEN 110 ELSE POKE X,A*2 : REM * RHYTHMS
100 X = X + 4 : GOTO 90 : REM * NOTICE RHYTHM MULTIPLIER ABOVE
110 RESTORE : REM * THIS GETS THE DATA POINTER BACK TO START
120 X = 24577 : REM * THESE LOCATIONS USED FOR SUBTLE DURATIONS
130 READ A : IF A = 999 THEN 150 ELSE POKE X,1 : REM * RHYTHMS
140 X = X + 4 : GOTO 130 : REM * NO CHANGE IN LSB TIMING ABOVE
150 POKE X,0 : POKE X+4,0 : POKE X+8,0 : REM * MUSIC END CODE
160 OUT254,3 : POKE 16526,96 : POKE 16527,143 : M=USR(0) : CLEAR
170 @LOAD4 : REM * PLAY NORM SPEED, USR CALL, CLEAR, LOAD NEXT
180 DATA0,0,0,0,0,0,0,22,26,30,30,30,30,34,34,34,0,0,0,0
190 DATA20,16,13,14,14,14,14,14,15,15,15,0,0,0,0,0,20,18,18
200 DATA18,18,24,0,24,0,24,0,24,0,24,24,24,0,24,0,24,24,0,24,0
210 DATA24,24,0,24,0,24,24,24,24,0,24,0,24,24,0,24,0,24,0,24,0
220 DATA22,22,18,18,16,16,14,14,13,13,12,13,13,13,14,19,27,19
230 DATA13,13,14,19,27,19,13,0,16,0,20,0,22,0,24,0,24,0,24,0,24
240 DATA0,20,16,13,14,14,20,16,13,0,16,0,16,0,16,12,12,12,13
250 DATA15,15,15,15,18,21,21,21,21,16,16,0,16,0,16,16,0,16,0,16
260 DATA17,15 : REM * end of page one for voice one
270 DATA20,21,24,26,0,0,0,0,0,22,26,30,30,30,30,34,34,34,0,0
280 DATA0,24,0,24,24,24,24,0,24,0,24,24,0,24,0,24,0,0,20,18
290 DATA18,18,24,0,24,0,24,0,21,0,22,0,18,0,30,30,32,0,24,0
300 DATA24,0,24,0,24,0,26,21,16,16,16,16,16,16,16,16,16,999
310 REM * THIS IS THE END OF THE PITCH SEQUENCE FOR VOICE ONE
320 DATA24,0,24,0,24,0,24,0,24,24,24,0,24,0,24,24,0,24,0,24,0
330 DATA24,0,24,24,24,24,0,24,0,24,24,0,24,0,24,24,0,24,0,24,26
340 DATA0,26,0,27,0,27,0,27,0,40,0,22,26,30,30,30,30,30,34,34
350 DATA34,0,0,0,0,0,0,20,16,13,14,14,14,14,14,15,15,15,0,0
360 DATA0,27,30,32,36,40,20,18,36,40,20,18,36,40,20,18,18,36
370 DATA36,40,20,18,18,36,36,40,0,30,0,26,0,32,0,27,0,27,0,30
380 DATA0,40,22,22,22,26,30,34,34,34,0,24,0,24,0,24,24,0,24
390 DATA0,24,24,0,24,0,24,24,0,24,0,24,26,26,26,22,30,30,30
400 DATA30,32,32,32 : REM * end of page one for voice two
410 DATA32,32,32,0,0,24,0,24,0,24,24,24,0,24,0,24,24,0,24,0,24
420 DATA0,20,16,13,14,14,14,14,15,15,15,0,0,0,0,24,0,26,26
430 DATA0,26,0,27,0,27,0,30,0,30,0,32,32,32,32,0,0,0,34,0
440 DATA34,0,34,0,34,0,32,28,24,24,26,24,26,24,26,26,26,999
450 REM * this is the conclusion of the score for voice two
460 DATA10,2,2,2,10,2,2,2,4,4,2,2,2,2,8,2,2,2,10,2,2,2,3,3,3
470 DATA2,2,2,8,2,2,2,8,2,2,2,8,2,2,2,2,2,2,10,2,2,2,10,2,2,2
480 DATA4,4,2,2,2,2,8,2,2,2,2,8,2,2,2,2,3,3,3,2,2,2,2,8,2,2,2,2
490 DATA10,2,2,2,4,4,4,4,8,8,8,8,8,8,8,8,4,4,4,4,8,8,4,4,4
500 DATA4,10,2,2,2,10,2,2,2,10,2,2,2,10,2,2,2,3,3,3,8,8,3,3,3
510 DATA8,10,2,2,2,8,2,2,2,8,2,2,2,8,2,2,2,2,8,2,2,2,2,8,2,2,2,8
520 DATA2,2,2,8,8,8 : REM * end of page one for durations
530 DATA8,8,16,8,8,10,2,2,2,4,4,2,2,2,8,2,2,2,10,2,2,2,3,3
540 DATA3,2,2,2,2,8,2,2,2,10,2,2,2,8,2,2,2,10,2,2,2,10,2,2
550 DATA2,10,2,2,2,2,2,4,8,10,2,2,2,10,2,2,2,4,4,8,8,1,1,1,1,3
560 DATA6,20,999 : REM * end of score for durations; end score

```

Listing 7-1. Examples of music programs.

## More Music

Naturally, there is much more to the creation of music by computer than the mere sounding of tones; the appearance of commercial music-generation peripherals for the TRS-80 attests to that. But though they may be well-designed pieces of electronics, composers and others serious about producing listenable music tend to look for richer sounds and more flexible ways to change that sound.

There are several ways to do this. The simplest is the tone generator with a tempered scale, capable of producing chordal sounds (consisting of three or more pitches sounding simultaneously). The tone color is limited to a fixed palette, and they are suitable for making elementary music. I refer to these generically as 'organs' because their tone colors usually give the illusion of organ stops. Most TRS-80 compatible music boards are simplified variations on the basic electronic organ, using integrated circuit chips like the General Instrument AY-3-8910 and the Texas Instrument SN76489.

A second group of electronic instruments creates music by constructing waveforms from a pre-assigned table of values. Thus, tone color, pitch, and volume can be altered by the selection of parameters in the table. This is digital synthesis in an elementary form, just one step beyond the simple software tone generation presented earlier in this Chapter. The TRS-80 is not capable of operating either with enough speed or electronic flexibility for digital synthesis.

Peripheral to the digital synthesis of tones is the electronic creation of vocal sounds. Texas Instruments, Votrax and other integrated circuits use a complex algorithm called 'linear predictive coding' to select from a known subset of human vocal sounds. Although somewhat convincing voices can be produced this way, mere intelligibility is the least significant criterion in music. Until (and if) predictive devices are developed for a wide array of musical sound, they have no application for producing music.

The most popular electronic music makers for more than a decade have been the analog synthesizers. Traditional oscillators create a sound which can be mutated and transformed until its color is right. These synthesizers were never conceived in computer-compatible terms, but companies such as PAIA have for the past few years offered hybrid analog-digital systems where the computer is used as a super-sequencer,

keeping the notes in order for storage and playback.

There are several reasons why I suggest interfacing analog synthesizers to the TRS-80. First, analog synthesizers are cheap. Small, capable machines can be picked up for the cost of a TRS-80, and kits are sold by PAIA and others for less than \$100. Surplus synthesizers are also available (*Moog, Buchla, PBI, Putney*, and others of early 1970's vintage) at low cost. All of these will take on new musical life when interfaced to the TRS-80.

Second, if you are a performing musician, it's likely you're looking for a musical instrument,

and it's in that area where the analog synthesizer is still champion. They are performance instruments, not electronic widgets. And with a computer interface, they remain stand-alone performance instruments, but with computer assistance where it is wanted.

Figure 7-7 presents the circuit for a 2- to 32-voice analog synthesizer interface. The basic circuit contains a data-line buffer (Z1), a buffer for the most used addresses (Z4), a port decoder (Z2/Z3), and a voice-pair selector (Z2/Z3). In a fully expanded system, sixty-four ports (port 64 to port 127) are used to provide thirty-two voltage outputs, sixty-four envelopes, with sixty-four additional control lines.

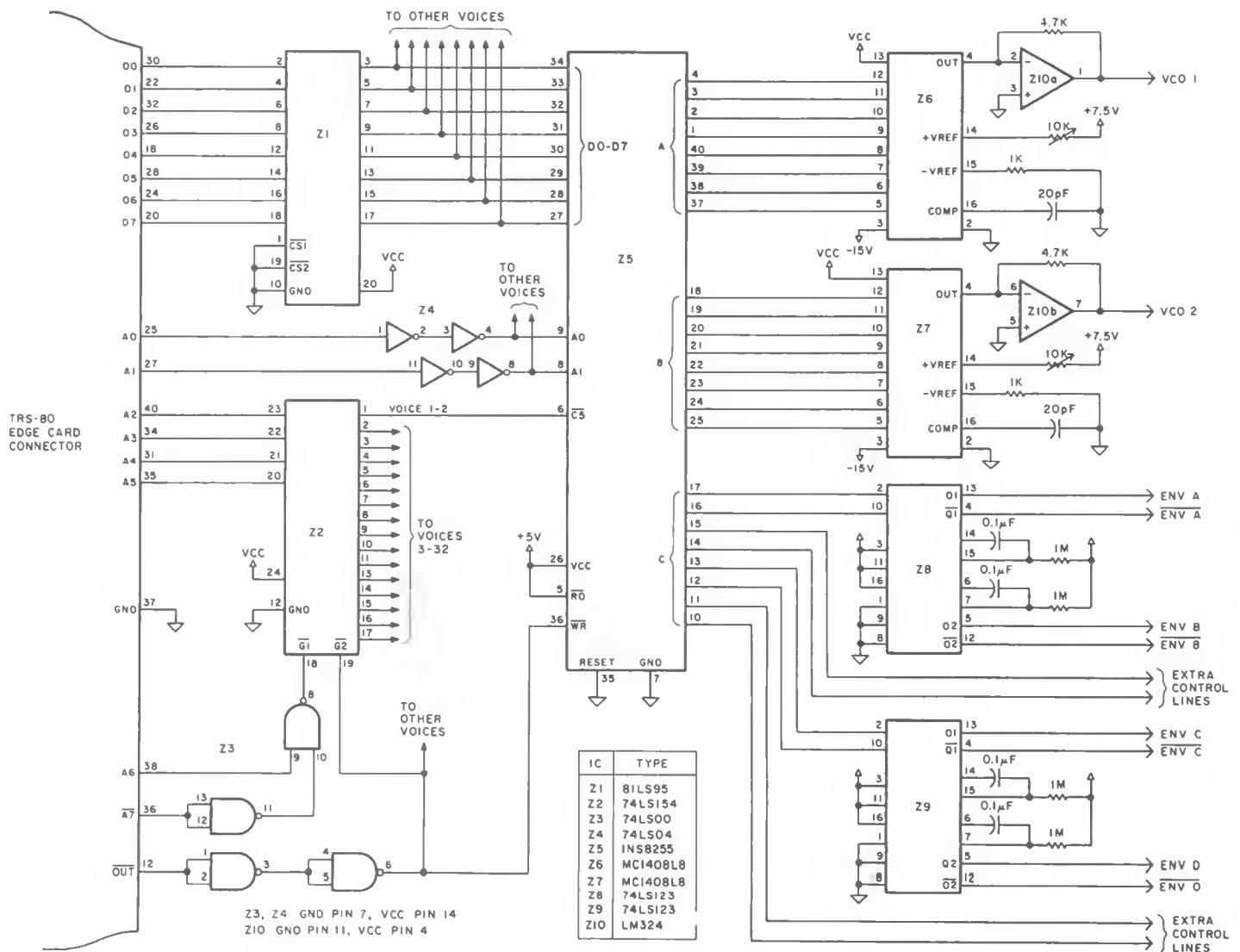


Figure 7-7. TRS-80 to voltage-controlled synthesizer interface.

```

10 OUT 254,2 : CLS : PRINT"READING.." : REM OUT254,2 = HISPEED
20 X = 24578 : REM * THIS IS MEM. POSITION OF VOICE NUMBER ONE
30 READ A : IF A = 999 THEN 50 ELSE POKE X,A : REM * PITCH POKE
40 X = X + 4 : GOTO 30 : REM * VOICE APPEARS EVERY 4 MEM LOC'NS
50 X = 24579 : REM * THIS IS MEM. POSITION OF VOICE NUMBER TWO
60 READ A : IF A = 999 THEN 80 ELSE POKE X,A : REM * PITCH POKE
70 X = X + 4 : GOTO 60 : REM * VOICE APPEARS EVERY 4 MEM LOC'NS
80 X = 24576 : REM * THIS IS THE BEGINNING VALUE FOR DURATIONS
90 READ A : IF A = 999 THEN 110 ELSE POKE X,A*2 : REM * RHYTHMS
100 X = X + 4 : GOTO 80 : REM * NOTICE RHYTHM MULTIPLIER ABOVE
110 RESTORE : REM * THIS GETS THE DATA POINTER BACK TO START
120 X = 24577 : REM * THESE LOCATIONS USED FOR SUBTLE DURATIONS
130 READ A : IF A = 999 THEN 150 ELSE POKE X,1 : REM * RHYTHMS
140 X = X + 4 : GOTO 130 : REM * NO CHANGE IN LSB TIMING ABOVE
150 POKE X,0 : POKE X+4,0 : POKE X+8,0 : REM * MUSIC END CODE
160 OUT254,3 : POKE 16526,96 : POKE 16527,143 : M=USR(0) : CLEAR
170 @LQA04 : REM * PLAY NORM SPEED, USR CALL, CLEAR, LQA0 NEXT
180 DATA0,0,0,0,0,0,0,0,22,26,30,30,30,30,30,34,34,0,0,0,0,0
190 DATA20,16,13,14,14,14,14,14,15,15,15,0,0,0,0,0,20,18,18
200 DATA18,18,24,0,24,0,24,0,24,0,24,0,24,0,24,0,24,0,24,0
210 DATA24,24,0,24,0,24,0,24,0,24,0,24,0,24,0,24,0,24,0
220 DATA22,22,18,18,16,16,14,14,13,13,12,13,13,13,14,19,27,19
230 DATA13,13,14,19,27,19,13,0,16,0,20,0,22,0,24,0,24,0,24,0
240 DATA24,0,20,16,13,14,14,20,16,13,0,16,0,16,0,16,12,12,12
250 DATA13,15,15,15,15,18,21,21,21,21,16,16,0,16,0,16,0,16,0
260 DATA16,17,15 : REM * END OF PAGE ONE FOR VOICE ONE
270 DATA20,21,24,26,0,0,0,0,0,22,26,30,30,30,30,34,34,0,0,0
280 DATA0,24,0,24,24,24,24,0,24,0,24,24,0,24,0,24,0,0,20,18
290 DATA18,18,18,24,0,24,0,24,0,24,0,21,0,22,0,18,0,30,30,32,0,24,0
300 DATA24,0,24,0,24,0,26,21,16,16,16,16,16,16,16,16,999
310 REM END OF PAGE TWO FOR VOICE ONE
320 DATA24,0,24,0,24,0,24,0,24,24,24,0,24,0,24,0,24,0,24,0
330 DATA24,0,24,24,24,24,0,24,0,24,24,0,24,0,24,24,0,24,0,26
340 DATA26,0,26,0,27,0,27,0,27,0,40,0,22,26,30,30,30,30,30,34
350 DATA34,34,0,0,0,0,0,0,20,16,13,14,14,14,14,15,15,16,15
360 DATA0,0,0,0,27,30,32,36,40,20,18,36,40,20,18,36,40,20,18,18
370 DATA36,36,40,20,18,18,36,36,40,0,30,0,26,0,32,0,27,0,27,0
380 DATA30,0,40,0,22,22,22,26,30,34,34,34,0,24,0,24,0,24,24,0
390 DATA24,0,24,24,0,24,0,24,24,0,24,0,24,26,26,26,22,30,30
400 DATA30,30,32,32,32 : REM * END OF PAGE ONE FOR VOICE TWO
410 DATA32,32,32,0,0,24,0,24,0,24,24,24,0,24,0,24,24,0,24,0
420 DATA0,0,0,20,16,13,14,14,14,14,14,15,15,15,0,0,24,0,26,26
430 DATA0,26,0,27,0,27,0,30,0,30,0,32,32,32,32,32,0,0,0,34,0,34
440 DATA0,34,0,34,0,32,28,24,24,26,24,26,24,26,28,26,999
450 REM * END OF PAGE TWO FOR VOICE TWO
460 DATA10,2,2,2,10,2,2,2,4,4,2,2,2,8,2,2,2,10,2,2,2,3,3,3
470 DATA2,2,2,2,8,2,2,2,2,2,2,2,8,2,2,2,2,10,2,2,2,10,2,2,2
480 DATA4,4,2,2,2,8,2,2,2,2,2,2,8,2,2,2,2,3,3,2,2,2,8,2,2,2
490 DATA10,2,2,2,4,4,4,8,8,8,8,8,8,8,8,8,4,4,4,8,8,4,4,4
500 DATA4,10,2,2,2,10,2,2,2,10,2,2,2,10,2,2,2,3,3,8,8,3,3,3
510 DATA8,10,2,2,2,8,2,2,2,2,2,2,8,2,2,2,2,8,2,2,2,2,8,2
520 DATA2,2,2,8,8,8 : REM * END OF PAGE ONE OF RHYTHMS
530 DATA8,8,16,8,8,10,2,2,2,4,4,2,2,2,8,2,2,2,10,2,2,2,3,3
540 DATA3,2,2,2,8,2,2,2,10,2,2,2,8,2,2,2,10,2,2,2,10,2,2
550 DATA2,10,2,2,2,2,2,4,8,10,2,2,2,10,2,2,2,4,8,8,1,1,1,3
560 DATA6,20,999 : REM * END OF RHYTHMS PAGE TWO AND END PIECE

```

Each pair of voices is managed by programmable peripheral interface Z5, which provides two 8-bit outputs to digital-to-analog converters Z6 and Z7, offering one-part-in-256 accuracy. Remaining from Z5 are eight control lines for envelope, etc. Optionally, six of these remaining control outputs can be used for voltage control of volume or filtering (see Figure 7-7), which can be used with less resolution (one part in 64) than that needed for pitch control.

Figure 7-8 is the power supply, which is similar to others in this book with the exception of the LM340-8, providing eight volts as a reference to the digital-to-analog converters.

The output of the D/A converters Z6 and Z7 is in the range of 0 volts to 4.98 volts, which should be more than adequate to drive most synthesizers across their full range. The envelope triggers provided by Z8 and Z9, however, may need tweaking depending on the type of synthesizer you are using. Some synthos require a negative-going pulse, and others need a positive-going one; likewise, some synthos trigger on the rising or falling *edge* of the wave, while other envelopes will sustain as long as the level of the trigger signal remains high or low. Thus, the resistance and capacitance values given for Z8 and Z9 may have to be changed for sustained envelope triggers, or Z8 and Z9 might be eliminated completely if the envelope is edge-triggered. If you don't have specs on your synthesizer, build the trigger circuit up, then down, until it works for your machine.

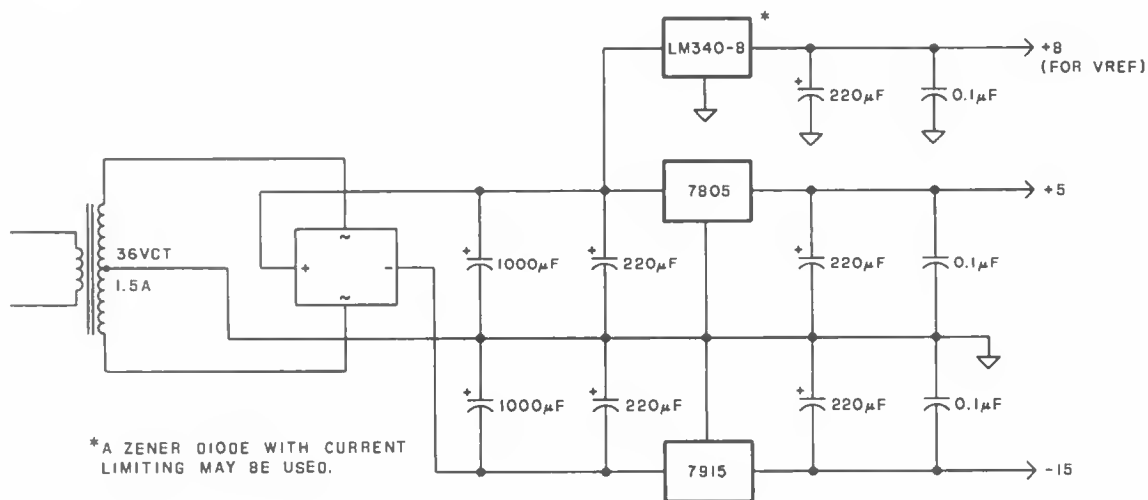


Figure 7-8. Power supply for the synthesizer interface.

```

10 OUT254,2 : CLS : PRINT"READING.." : REM * OUT254,2 = HISPEED
20 X = 24578 : REM * THIS IS MEM. POSITION OF VOICE NUMBER ONE
30 READ A : IF A = 999 THEN 50 ELSE POKE X,A : REM * EXIT @ 999
40 X = X + 4 : GOTO 30 : REM * VOICE APPEARS EVERY 4 MEM LOC'NS
50 X = 24579 : REM * THIS IS MEM. POSITION OF VOICE NUMBER TWO
60 READ A : IF A = 999 THEN 80 ELSE POKE X,A : REM * PITCH POKE
70 X = X + 4 : GOTO 60 : REM * VOICE APPEARS EVERY 4 MEM LOC'NS
80 X = 24576 : REM * THIS IS THE BEGINNING VALUE FOR DURATIONS
90 READ A : IF A = 999 THEN 110 ELSE POKE X,A*2 : REM * RHYTHMS
100 X = X + 4 : GOTO 90 : REM * NOTICE RHYTHM MULTIPLIER ABOVE
110 RESTORE : REM * THIS GETS THE DATA POINTER BACK TO START
120 X = 24577 : REM * THESE LOCATIONS USED FOR SUBTLE DURATIONS
130 READ A : IF A = 999 THEN 150 ELSE POKE X,1 : REM * RHYTHMS
140 X = X + 4 : GOTO 130 : REM * NO CHANGE IN LSB TIMING ABOVE
150 POKE X,0 : POKE X+4,0 : POKE X+8,0 : REM * MUSIC ENO CODE
160 OUT254,3 : POKE 16526,96 : POKE 16527,143 : M=USR(0) : CLEAR
170 @LQAO6 : REM * PLAY NORM SPEED, USR CALL, CLEAR, LOAD NEXT
180 DATA0,0,0,0,0,0,0,0,25,29,31,29,3,0,0,0,20,16,13,14,14,14
190 DATA14,14,15,15,15,0,0,0,0,0,20,18,18,18,18,24,0,24,0,24
200 DATA0,24,0,24,24,24,0,24,0,24,24,0,24,0,24,0,24,0,24,24
210 DATA24,24,0,24,0,24,24,0,24,0,24,0,24,0,22,22,18,16,16
220 DATA14,14,13,13,12,13,13,13,14,19,27,19,13,13,14,19,27,19
230 DATA13,0,16,0,20,0,22,0,24,0,24,0,24,0,12,12,12,12,13
240 DATA15,15,15,15,18,21,21,21,16,16,0,16,0,16,0,16,0,16,0,16
250 DATA17,15 : REM * ENO OF PAGE ONE FOR VOICE ONE
260 DATA3,140,3,22,26,30,30,30,30,30,34,34,34,0,0,0,24,0,24,24
270 DATA24,24,0,24,0,24,24,0,24,0,24,0,0,20,18,18,18,24,0
280 DATA24,0,24,0,21,0,22,0,18,0,30,30,32,0,24,0,24,0,24,0
290 DATA26,21,16,16,16,16,16,16,16,16,999 : REM * ENO P2V1
300 DATA24,0,24,0,23,3,135,1,1,42,0,24,0,24,0,24,24,24,0,24
310 DATA0,24,24,0,24,0,24,24,0,24,0,26,26,0,26,0,27,0,27,0,27,0
320 DATA40,0,22,26,30,30,30,30,30,34,34,34,0,0,0,0,20,16,13
330 DATA14,14,14,14,14,15,15,15,15,0,0,0,27,30,32,36,40,20
340 DATA18,36,40,20,18,12,140,140,13,3,40,20,18,18,36,36,40,0
350 DATA30,0,26,0,32,0,27,0,27,0,30,0,40,0,22,22,22,26,30,34,34
360 DATA34,0,24,0,24,0,24,24,0,24,0,24,24,0,24,0,24,0,24,0,24,0
370 DATA24,26,26,26,26,22,30,30,30,30,32,32,32 : REM * ENO P1V2
380 DATA32,32,32,0,0,24,0,24,0,24,24,24,0,24,0,24,0,24,0,24,0
390 DATA0,0,0,20,16,13,14,14,14,14,15,15,15,0,0,0,24,0,26,26
400 DATA0,26,0,27,0,27,0,30,0,30,0,32,32,32,32,0,0,0,34,0,34
410 DATA0,34,0,34,0,32,28,24,24,26,24,26,24,26,28,26,999
420 REM * ENO OF PAGE TWO FOR VOICE TWO
430 DATA10,2,2,2,10,2,2,2,4,4,2,2,2,8,2,2,2,10,2,2,2,3,3,3
440 DATA2,2,2,8,2,2,2,2,8,2,2,2,2,8,2,2,2,10,2,2,2,10,2,2,2
450 DATA4,4,2,2,2,8,2,2,2,2,8,2,2,2,2,3,3,3,2,2,2,8,2,2,2,2
460 DATA10,2,2,2,4,4,4,8,8,8,8,8,8,8,8,4,4,4,4,8,3,11,5,4
470 DATA10,2,2,2,10,2,1,1,10,2,2,2,10,2,2,2,3,3,3,8,3,3,3,8
480 DATA10,2,2,2,10,2,2,2,2,8,2,2,2,2,8,2,2,2,2,8,2,2,2,8,2,2
490 DATA2,2,8,8,8 : REM * ENO OF PAGE ONE OF RHYTHMS
500 DATA8,8,16,8,10,2,2,2,4,4,2,2,2,8,2,2,2,2,10,2,2,2,3,3
510 DATA3,2,2,2,2,8,2,2,2,2,10,2,2,2,8,2,2,2,2,10,2,2,2,10,2,2
520 DATA2,10,2,2,3,13,5,9,2,2,2,12,10,2,2,2,4,4,8,8,1,1,1,1,3,6
530 DATA20,999 : REM * ENO PAGE TWO OF RHYTHMS AND ENO PIECE

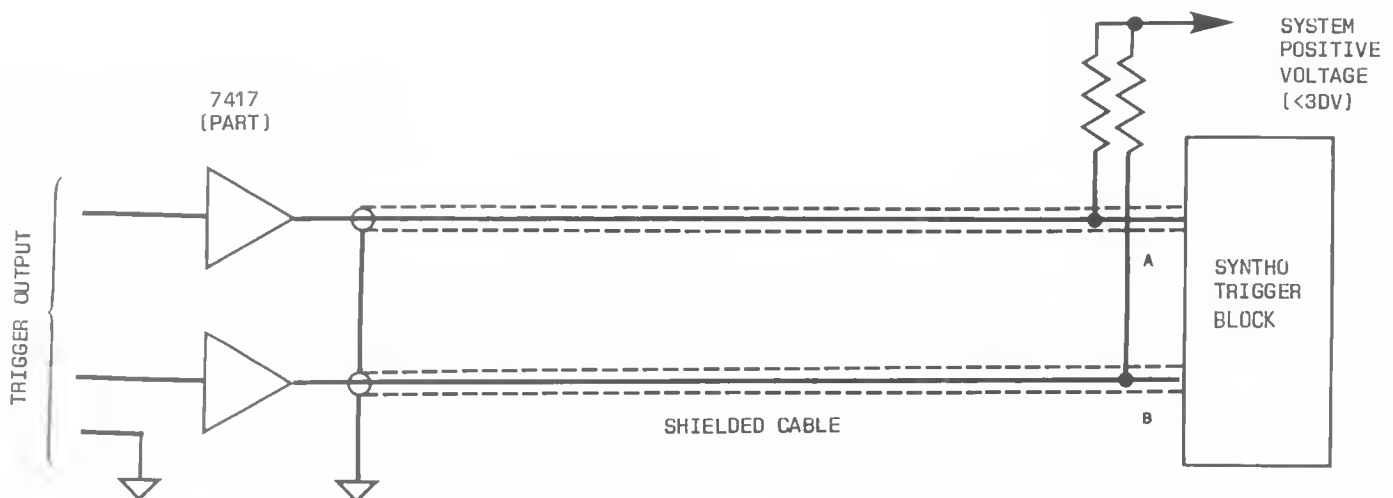
```

Synthesizer Interface Port Addressing

Port Number (Decimal)	Port Number (Hex)	Port Function
64	40	Pitch Control Voice 1
65	41	Pitch Control Voice 2
66	42	Envelopes 1 a/b and 2 a/b Extre Lines 1 a/b and 2 a/b
67	43	Port Control Voices 1 - 2
68	44	Pitch Control Voice 3
69	45	Pitch Control Voice 4
70	46	Envelopes 3 a/b and 4 a/b Extre Lines 3 a/b and 4 a/b
71	47	Port Control Voices 3 - 4
72	48	Pitch Control Voice 5
73	49	Pitch Control Voice 6
74	4A	Envelopes 5 a/b and 6 a/b Extre Lines 5 a/b and 6 a/b
75	4B	Port Control Voices 5 - 6
76	4C	Pitch Control Voice 7
77	4D	Pitch Control Voice 8
78	4E	Envelopes 7 a/b and 8 a/b Extre Lines 7 a/b and 8 a/b
78	4F	Port Control Voices 7 - 8
.	.	.
.	.	.
.	.	.
.	.	.
124	7C	Pitch Control Voice 31
125	7D	Pitch Control Voice 32
126	7E	Envelopes 31 a/b and 32 a/b Extre Lines 31 a/b and 32 a/b
127	7F	Port Control Voices 31 - 32

Table 7-1. Synthesizer interface port addressing.

Using the synthesizer interface is straightforward. Plug the interface into the TRS-80 expansion connector, and run shielded microphone cable from the voltage outputs of the interface to the voltage inputs (marked 'control in', 'voltage in', 'VCO control', 'external in', or something similar) on the synthesizer. Then run either parallel speaker wire or shielded cable from the interface envelope (positive-going edge) to the synthesizer's envelope inputs. The integrated circuits running from the interface are not balanced for long lines; if you plan to use more than a dozen feet of cable, place 7417 open-collector buffers at each control output from the interface, so:



Once the connections are made, run the following few lines, an envelope test:

```

10 OUT254,2:CLS:PRINT"READING..."
12 X=24578
15 READA:IFA=889THEN30ELSEPOKEX,A:X=X+4:GOTO15
30 X=24578
35 READA:IFA=888THEN50ELSEPOKEX,A:X=X+4:GOTO35
50 X=24578
55 READA:IFA=887THEN80ELSEPOKEX,A*X=X+4:GOTO55
80 RESTORE:X=24577
85 READA:IFA=999THEN87ELSEPOKEX,1:X=X+4:GOTO80
97 OUT254,3:POKEX+4,0:POKEX+8,0:POKE16526,86:POKE16527,143:M=USR(0):CLEAR
98 *LOAD3
100 DATA20,20,20,20,20,20,20,18,20,19,17,17,17,17,17,15
110 DATA24,24,21,27,28,27,27,27,30,27,30,27,0,0,20,21,20,19,17
120 DATA16,16,27,24,40,48,48,44,40,38,30,28,26,18,18,14,13
130 DATA11,8,8,8,12,11,12,10,21,21,27,27,28,27,28,24
140 DATA28,28,28,27,24,18,17,17,20,13,13,18,20,21,24,22,22,22,20
150 DATA18,18,36,27,30,30,28,36,18,18,20,13,13,22,22,24,24,17,15,15
160 DATA14,27,24,21,20,21,18,14,13,11,10,13,18,24
170 DATA32,32,32,32,32,32,32,32,32,32,32,32,30,27,40,38,38,28,26,24,24
180 DATA20,20,20,20,20,20,20,18,20,18,17,17,17,17,15
190 DATA24,24,21,27,26,27,27,27,30,27,27,24,22,32,32,24,22
200 DATA34,34,32,42,42,38,38,34,30,24,22,20,14,14
210 DATA13,13,14,16,18,18,18,17,16,20,22,28,27,27,27,30,24,18
220 DATA17,17,17,17,18,16,14,13,18,22,28,27,27,27,34,28,28,22,22,22
230 DATA24,24,20,30,38,38,40,38,42,42,42,44,54,68,68,68,68,48,52,44,40,34,34,38
240 REM END OF PAGE 1
250 DATA38,44,32,32,34,40,28,28,30,36,28,27,32,22,24,30,32,36,38,42,40,36
260 DATA40,40,38,44,32,32,34,40,28,28,38,32,27,27,24,21
270 DATA20,20,20,20,20,20,20,18,20,18,17,17,17,17,15,24,26,27,30,32,34,36,0
280 DATA0,0,0,0,0,0,0,0,999
290 REM END OF PAGE 2
300 DATA60,52,48,40,34,38,42,50,52,48,48,40,34,36,42,42
310 DATA80,52,48,48,40,34,38,42,42,60,52,52,48,40,34,34,38,38,42
320 DATA54,52,42,38,30,32,38,54,54,52,52,42,36,30,32,38,38
330 DATA54,52,42,38,40,30,32,38,54,52,42,36,36,30,32,38
340 DATA48,44,38,32,32,27,27,28,28,34,48,44,44,38,38,32,32,27,28,34,34
350 DATA48,48,44,44,38,32,32,27,28,34,48,44,44,38,38,32,32,27,28,34
360 DATA36,38,38,38,52,52,62,64,64,68,68,68,68
370 DATA68,60,60,54,42,48,40,44,52,40,54,40,57,72,78,78,44,52,57,64,40,54,57
380 DATA60,57,48,40,34,36,42,60,57,48,48,40,34,36,42,42
390 DATA60,57,48,48,40,34,36,42,42,64,68,76,76,48,36,40,40
400 DATA40,57,54,48,60,64,42,52,52,52,48,54,60,68
410 DATA78,72,60,52,42,44,54,54,76,78,72,72,80,52,42,44,44,54
420 DATA40,42,52,50,60,72,72,68,68,57,57,40,42,52,52,60,60,72,68,57
430 DATA38,38,38,44,44,54,54,84,84,80,52,52,58,56,36,44,54,64,84,80,60,52,52
440 REM END OF PAGE 1
450 DATA54,54,54,72,48,48,48,84,42,42,40,38,38,38,52,52,34,34,34,44,44
460 DATA30,28,27,27,27,38,24,24,24,32,21,20,18,28,42,68,68
470 DATA60,57,48,40,34,38,42,80,57,48,48,40,34,36,42,42
480 DATA60,57,48,44,42,0,0,0,40,0,0,0,0,0,0,0,0,0,0,0,0,999
490 REM END OF PAGE 2
500 DATAB,8,8,8,8,8,8,8,8,4,4,8,8,8,4,4
510 DATAB,8,8,4,4,8,8,8,4,4,8,4,4,8,8,4,4,4,8
520 DATAB,8,8,8,8,8,8,4,4,4,8,8,8,8,4,4
530 DATAB,8,8,8,4,4,8,8,8,8,8,4,4,8,8,8
540 DATAB,8,8,8,4,4,4,4,4,8,8,4,4,4,4,4,8,8,4,4
550 DATA4,4,4,4,8,4,4,8,8,8,8,4,4,4,4,4,4,4,8,8
560 DATAB,8,8,8,8,8,8,8,8,8,8,8,8,8,8
570 DATA4,4,4,4,4,4,4,4,4,4,4,4,4,8,8,8,8,4,4,4,8,4,4
580 DATAB,8,8,8,8,8,8,8,8,8,4,4,8,8,8,4,4
590 DATAB,8,4,4,8,8,8,4,4,8,8,4,4,8,8,8
600 DATAB,8,8,8,8,8,8,8,8,8,8,8,8,8,8
610 DATAB,8,8,8,4,4,4,4,4,4,4,4,4,8,8,8,4,4,8,8
620 DATAB,8,8,8,4,4,4,4,4,4,4,4,4,8,8,8,8,8
630 DATAB,4,4,4,4,4,4,4,4,8,4,4,4,4,8,8,4,4,4,4,4,4,4
640 REM END OF PAGE
650 DATA4,4,8,8,4,4,8,8,4,4,8,8,4,4,4,4,4,4,4,4
660 DATAB,8,4,4,8,8,4,4,8,8,8,8,8,8,8,4,4
670 DATAB,8,8,8,8,8,8,8,8,8,4,4,8,8,8,8,8,4,4,8
680 DATAB,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8

```

```

10 OUT 67,128 : REM SET 8255 PORT
20 A$=INKEY$ : REM SCAN KEYBOARD
30 IF A$="" THEN 20 : REM LOOP IF NO KEY
40 OUT 66,1 : REM ENVELOPE ON SHOT
50 OUT 66,0 : REM ENVELOPE OFF SHOT
60 GOTO 20 : REM LOOP AS NEEDED

```

The envelope should be triggered each time you touch a key on the TRS-80. If the envelope does not trigger, move the interface connection to the negative-going envelope, and try again. If the envelope is still not working, increase the values for C3 or R5, which are found at pins 14 and 15 of Z8. This will lengthen the trigger cycle, and should handle any synthesizer input. Again, try both the positive and negative envelopes.

Now disconnect the envelope trigger, and patch the envelope out of the synthesizer. Try the following program which creates a series of fast-rising whoops if the digital-to-analog converter is working properly:

```

10 OUT 67,128 : REM SET UP 8255 PORT
20 INPUT"TIME DELAY":N : REM GET INTERNOTE DELAY
30 FOR X = 0 TO 255 : REM SET TO RUN ALL NOTES
40 OUT 64,X : REM SEND VOLTS TO SYNTHO
50 FOR Y = 1 TO N : REM SET UP TIMING LOOP
60 NEXT : NEXT : REM TIME, GET NEXT NOTE
70 GOTO 20 : REM BACK FOR NEXT DELAY

```

Each time you run through the program to the INPUT statement, increase the value for the time delay; the whoops will slow until you can hear a series of discrete pitches. If you have a two-voice synthesizer (or to test the second voice of the interface), connect the voltage output from voice 2, and use OUT 71,128 to set the port and OUT 68,X for the data.

If the pitches fall instead of rise, then your synthesizer responds to a higher voltage as a *lower* pitch, and any music routines you write will have to take this into account. If this is the case, before you run change line 60 to read:

```
60 OUT 66,32 : OUT 66,0 : OUT 64,(255-ML(1,PH))
```

Finally, run Listing 7-4, a rendition of keyboard prelude 23 from Johann Sebastian Bach's Well-Tempered Clavier.

```

10 GOSUB 650 : REM * THE TUNING SECTION IS RETRIEVED FIRST
20 DEFINT V, A, B, C, O, T, X, Y, P, N : REM * INTEGERS ALL
30 V1 = 64 : REM * VOICE ONE USES OUTPUT LOCATION 64
40 V2 = 65 : REM * VOICE TWO USES OUTPUT LOCATION 65
50 EN = 66 : REM * ENVELOPE TRIGGER USES OUTPUT LOCATION 66
60 TN = 32 : REM * TRIGGER TURNED ON WITH VALUE 32
70 TF = 0 : REM * TRIGGER TURNED OFF WITH VALUE 0
80 CT = 67 : REM * 8255 PIA CONTROL REGISTER AT LOCATION 67
90 PT = 128 : REM * PORT OUTPUT ONLY DEFINED BY VALUE 128
100 NN = 200 : REM * NUMBER OF NOTES DEFINED BY VALUE 200
110 DIM V1(2,500) : REM * ARRAY OF NOTES AND THEIR DURATIONS
120 DIM V2(2,500) : REM * ARRAY OF NOTES AND THEIR DURATIONS
130 X=1 : REM * TEMPORARY COUNTER ONLY
140 READ V1(1,X) : REM * READ VOICE ONE
150 READ V1(2,X) : REM * READ DURATION ONE
160 IF V1(2,X) = 0 THEN 170 ELSE X = X + 1 : GOTO 140
170 X=1 : REM * TEMPORARY COUNTER ONLY
180 READ V2(1,X) : REM * READ VOICE TWO
190 READ V2(2,X) : REM * READ DURATION TWO
200 IF V2(2,X) = 0 THEN 210 ELSE X = X + 1 : GOTO 180
210 A=1 : B=1 : C=1 : D=1 : X=1 : Y=1 : Q=0 : R=2
220 OUT CT,PT : OUT V1,255 : OUT V2,255
230 G = V1(R,X) : H = V2(R,Y) : GOSUB 330 : GOSUB 350
240 G = G - A : IF G <= 0 THEN 250 ELSE 270
250 X = X + A : GOSUB 330 : G = V1(R,X)
260 GOTO 280
270 FOR W = 1 TO 14 : NEXT
280 H = H - A : IF H <= 0 THEN 290 ELSE 310
290 Y = Y + A : GOSUB 350 : H = V2(R,Y)
300 GOTO 320
310 FOR W = 1 TO 14 : NEXT
320 IF X > 200 THEN 210 ELSE 240
330 IF V1(A,X) = Q THEN 340 ELSE OUT EN,TN : OUT EN,TF :
OUT V1,V1(A,X) : RETURN
340 FOR W = 1 TO 2 : NEXT : RETURN

```

Listing Continued . . .

## Continued Listing

```

350 IF V2(A,Y) = Q THEN 360 ELSE OUT EN,TN : OUT EN,TF :
    OUT V2,V2(C,Y) : RETURN
360 FOR W = 1 TO 2 : NEXT : RETURN
370 DATA81,4,105,2,102,2,87,4,114,4,102,4,120,2,117,2,132,4,126
380 DATA4,90,4,96,6,93,2,96,4,123,2,117,2,123,4,132,9,139,2,111
390 DATA2,117,2,132,2,129,2,132,2,138,4,144,2,135,2,138,4,138,4
400 DATA108,4,123,4,87,4,117,4,93,4,95,2,90,2,87,2,84,2,87,12
410 DATA108,4,102,12,132,4,129,4,138,4,135,4,129,4,123,4,111,2
420 DATA120,2,123,4,87,4,108,4,75,2,81,2,87,4,108,2,105,2,108
430 DATA2,132,2,138,2,123,2,126,4,111,4,96,4,99,4,87,4,90,8,93
440 DATA8,102,8,105,8,108,4,132,4,93,4,138,4,81,4,105,2,102,2
450 DATA87,4,114,4,102,4,120,2,117,2,132,4,126,4,90,4,87,2,90,2
460 DATA96,2,102,2,96,2,102,2,105,2,87,2,93,2,129,2,132,2,126,2
470 DATA123,2,126,2,120,2,117,2,111,2,105,2,102,2,111,2,108,2
480 DATA111,2,120,2,117,2,111,4,138,2,117,2,123,4,121,2,105,2
490 DATA132,2,126,2,120,2,117,2,111,4,132,2,129,2,132,4,102,4
500 DATA81,4,87,2,90,2,86,2,98,2,96,2,117,2,111,2,108,2,111,4
510 DATA114,4,111,16,0,0,0,36,81,4,105,2,102,2,87,4,114,4,102
520 DATA4,120,2,117,2,132,4,126,4,90,4,96,6,93,2,96,4,123,2,117
530 DATA2,123,4,132,8,138,2,111,2,117,2,132,2,129,2,132,2,138,4
540 DATA144,2,135,2,138,4,138,4,108,4,123,4,87,4,117,4,93,4,96
550 DATA2,80,2,87,2,84,2,87,12,108,4,102,12,132,4,129,4,138,4
560 DATA135,4,129,4,123,4,111,2,120,2,123,4,87,4,105,4,75,2,81
570 DATA2,87,4,108,2,105,2,108,2,132,2,128,2,123,2,126,4,111,4
580 DATA88,4,99,4,87,4,90,8,93,8,102,8,105,8,108,4,132,4,93,4
590 DATA138,4,81,4,105,2,102,2,87,4,114,4,102,4,120,2,117,2,132
600 DATA4,126,4,90,4,87,2,90,2,96,2,102,2,96,2,102,2,105,2,87,2
610 DATA93,2,129,2,132,2,126,2,123,2,126,2,120,2,117,2,111,2
620 DATA105,2,102,2,111,2,108,2,111,2,120,2,117,2,111,4,138,2
630 DATA117,2,123,4,141,2,105,2,132,2,126,2,120,2,117,2,111,4
640 DATA132,2,128,2,132,16,0,0 : REM * END OF TWO VOICES
650 REM * THE TUNING SECTION PRODUCES OCTAVE PITCHES
660 OUT 67,128 : REM * SET UP THE 8255 PIA FOR ACTION
670 OUT 64,3 : OUT 65,3 : FOR N = 1 TO 1000 : NEXT
680 OUT 64,38 : OUT 65,38 : FOR N = 1 TO 1000 : NEXT
690 OUT 64,75 : OUT 65,75 : FOR N = 1 TO 1000 : NEXT
700 OUT 64,111 : OUT 65,111 : FOR N = 1 TO 1000 : NEXT
710 OUT 64,147 : OUT 65,147 : FOR N = 1 TO 1000 : NEXT
720 A$ = INKEY$ : IF A$ = "" THEN 670 ELSE RETURN

```

```

10 CL8 : PRINTCHR$(23) "-- GENERALIZED MUSIC PROGRAM --"
20 PRINT : PRINT : PRINT "DENNIS BATHORY KITSZ, MAY 1979"
30 FOR X = 1 TO 1000 : NEXT : CLEAR : CLS
40 PRINT"THESE ARE 639 NOTES IN EACH PRELUDE NO. 23" : Q = 939
50 GOTO 400 : REM * MOVE TO TUNING ROUTINE BEFORE PLAYING PIECE
60 DIM ML(2,0+50) : REM * EXTRA BYTES GET ASIDE FOR LONGER WORK
70 PRINT : PRINT"NOW READING";0;"NOTES FROM NOTE/RHYTHM DATA."
80 FOR PH = 1 TO 0 : REM * PH SPECIFIES PITCH & DURATION ARRAY
90 FOR NT = 1 TO 2 : REM * NT SPECIFIES THE TWO ARRAY ELEMENTS
100 READ ML(NT,PH) : REM * EACH PITCH AND RHYTHM FILLS ARRAY
110 NEXT NT,PH : REM * READ ENTIRE ARRAY OF 639 NOTES & RHYTHMS
120 INPUT "PRESS ENTER TO START PIECE";Z : REM * WAIT FOR START
130 FOR PH = 1 TO 0 : REM * READ ARRAY LOOP TO PLAY BACK MUSIC
140 OUT 55,32 : OUT 59,0 : REM * ENVELOPE TRIGGER ON THEN OFF
150 OUT 54,ML(1,PH) : REM * THEN SEND PITCH TO D/A CONVERTER
160 FOR X = 1 TO ML(2,PH) : NEXT : REM * ARRAY DURATION DELAY
170 NEXT PH : REM * AND CONTINUE UNTIL ALL NOTES ARE PLAYED OUT
180 INPUT "PRESS ENTER TO PLAY AGAIN";Z : REM * OPTIONAL REPEAT
190 GOTO 120 : REM * ANOTHER PIECE MAY BE ADDED AND RUN HERE
200 DATA88,31,80,31,117,31,90,31,66,31,80,31,111,31,80,31,88,
    31,80,31,117,31,80,31,54,31,102,31,126,31,102,31,60,31,
    81,31,105,31,81,31,54,31,81,31,102,31,81,31,80,31,81,31,
    105,31,81,31,45,31,80,31,117,31,80,31
210 DATA48,31,89,31,89,31,89,31,45,31,89,31,90,31,58,31,48,31,
    68,31,69,31,86,31,33,31,81,31,105,31,81,31,39,31,81,31,84,
    31,81,31,33,31,81,31,88,31,81,31,30,31,75,31,80,31,75,31,
    18,31,85,31,84,31,69,31
220 DATA89,31,81,31,90,31,81,31,54,31,88,31,81,31,88,31,45,31,
    54,31,88,31,54,31,33,31,45,31,54,31,45,31,24,31,45,31,51,
    31,57,31,60,31,89,31,89,31,75,31,89,31,60,31,66,31,69,31,
    75,31,81,31,87,31,80,31
230 DATA75,31,87,31,88,31,87,31,60,31,75,31,87,31,75,31,51,31,
    80,31,75,31,60,31,39,31,51,31,80,31,51,31,30,31,51,31,54,
    31,50,31,88,31,72,31,75,31,81,31,75,31,66,31,72,31,75,31,
    81,31,87,31,60,31,88,31
240 DATA81,31,80,31,102,31,90,31,68,31,81,31,90,31,81,31,54,31,
    88,31,81,31,88,31,45,31,54,31,85,31,54,31,33,31,54,31,81,
    31,54,31,24,31,54,31,68,31,54,31,30,31,54,31,75,31,54,53,
    19,31,54,31,84,31,54,31
250 DATA33,31,54,31,81,31,54,31,24,31,54,31,69,31,54,31,30,31,
    54,31,75,31,54,31,18,31,54,31,84,31,54,31,33,31,54,31,81,
    31,54,31,39,31,80,31,60,31,60,31,39,31,60,31,87,31,60,31,
    42,31,88,31,88,31,66,31
260 DATA45,31,88,31,80,31,66,31,8,31,88,31,102,31,88,31,15,31,
    88,31,88,31,88,31,81,31,75,31,105,31,75,31,54,31,75,31,
    102,31,75,31,18,31,81,31,111,31,81,31,24,31,81,31,105,31,
    81,31,80,31,87,31,117,31,87,31
270 DATA88,31,75,31,81,31,87,31,80,31,89,31,102,31,105,31,111,
    31,106,31,102,31,88,31,80,31,84,31,81,31,75,31,81,31,69,
    31,68,31,60,31,54,31,48,31,45,31,39,31,33,31,39,31,45,31,
    38,31,33,31,30,31,24,31,18,31
280 DATA15,31,75,31,105,31,75,31,18,31,75,31,102,31,75,31,9,31,
    89,31,102,31,88,31,15,31,89,31,88,31,58,31,19,31,88,31,88,
    31,88,31,30,31,89,31,80,31,68,31,33,31,80,31,90,31,60,31,
    38,31,80,31,87,31,80,31
290 DATA18,31,75,31,80,31,75,31,66,31,80,31,75,31,88,31,54,31,
    88,31,75,31,88,31,54,31,75,31,68,31,54,31,39,31,54,31,51,
    31,45,31,38,31,33,31,30,31,24,31,18,31,30,31,38,31,54,35,
    89,38,75,42,84,45,93,48
300 DATA8,250,45,15,54,15,89,15,81,15,99,15,105,15,117,15,128,

```

## Continued Listing

```

200,45,15,54,15,89,15,81,15,99,15,105,15,117,15,128,50,38,
    15,45,15,80,15,89,15,108,15,117,15,128,131,132,31,135,31,
    132,31,125,31,120,31,117,31,120,31
310 DATA126,31,123,31,117,31,111,31,108,31,111,31,117,31,111,
    31,108,31,102,31,89,31,102,31,108,31,102,31,88,31,80,31,
    84,31,81,31,75,31,88,31,63,31,80,31,54,31,48,31,45,31,38,
    31,38,31,30,34,24,37,18,40,12,44,8,48,3,250
320 DATA38,15,48,15,80,15,75,15,88,15,111,15,120,200,38,15,48,
    15,60,15,75,15,89,15,105,15,111,15,120,50,54,15,88,15,75,
    15,80,15,102,15,111,15,120,131,64,31,39,31,45,31,48,31,54,
    31,80,31,88,31,88,31
330 DATA69,31,80,31,54,31,80,31,88,31,89,31,75,31,81,31,75,31,
    89,31,89,31,89,31,75,31,81,31,84,31,80,31,84,31,81,31,75,
    31,81,31,84,31,80,31,88,31,102,31,105,31,111,31,117,31,
    120,34,128,37,132,40,138,44,125,48
340 DATA141,250,33,15,45,15,54,15,89,15,80,15,102,15,105,15,
    117,200,33,15,45,15,54,15,89,15,80,15,102,15,105,15,117,
    50,48,15,80,15,89,15,84,15,85,15,105,15,117,250,51,15,50,
    15,88,15,89,15,105,15,114,200,51,15,80,15,75,15,85,15,105
350 DATA15,111,50,54,15,89,15,75,15,84,15,80,15,102,15,111,250,
    102,15,111,31,102,31,80,31,84,31,80,31,102,31,111,31,102,
    31,111,31,102,31,80,31,84,31,80,31,102,31,111,31,129,31,
    125,31,120,31,114,31,111,31,105,31,102,31,88,31
360 DATA102,31,80,31,102,31,111,31,120,31,129,31,138,31,147,31,
    141,83,111,31,114,31,102,31,105,31,87,31,18,15,38,15,48,
    15,54,15,75,15,80,15,105,15,111,125,81,125,18,15,38,15,48,
    15,54,15,84,15,90,15,102,15,111,103,54,125,80,125
370 DATA33,15,45,15,54,15,93,15,81,15,80,15,105,250,81,31,80,
    31,81,31,89,31,89,31,89,31,81,31,80,31,81,31,81,31,81,31,
    58,31,69,31,58,31,81,31,80,31,108,31,89,31,80,31,81,31,72,
    31,85,31,54,31,81,31
380 DATA80,15,84,15,81,15,84,15,81,15,84,125,81,31,75,31,81,31,
    84,31,81,31,75,31,68,31,89,31,88,31,75,31,89,31,88,31,80,
    31,54,31,80,31,89,31,80,31,54,31,48,31,45,31,48,31,54,31,
    48,31,45,31,38,31,33,31,30,31
390 DATA33,31,45,31,54,31,45,31,45,31,54,31,89,31,54,31,54,31,
    89,31,81,31,89,31,88,31,81,31,80,31,81,31,81,31,80,31,105,
    31,80,31,80,31,105,31,117,33,105,35,105,37,117,39,128,42,
    117,45,141,150
400 REM * * * * *
410 PRINT"THIS IS THE TUNING SECTION -- OUTPUT IS 3, 38, 75."
420 PRINT"HOLD SPACE BAR TO BEGIN READING EACH PRELUDE DATA."
430 OUT 54,3 : FOR X = 1 TO 500 : NEXT : REM * OUTPUT LOW C
440 OUT 54,38 : FOR X = 1 TO 500 : NEXT : REM * NEXT OCTAVE
450 OUT 54,75 : FOR X = 1 TO 500 : NEXT : REM * NEXT OCTAVE
460 OUT 54,111 : FOR X = 1 TO 500 : NEXT : REM * NEXT OCTAVE
470 OUT 54,147 : FOR X = 1 TO 500 : NEXT : REM * TOP OCTAVE
480 A$ = INKEY$ : IF A$ = "" THEN 490 : REM * CHECK KEYBOARD
490 OUT 58,255 : REM * HOPE THAT VERY TOP NOTE IS INAUDIBLE
500 GOTO 80 : REM * NOW RETURN TO MAIN MUSIC READING ROUTINE

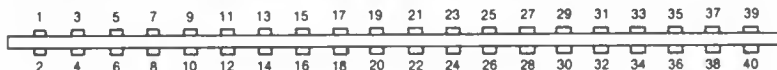
```

## Listing 7-4. Johann Sebastian Bach.



P/N	SIGNAL NAME	DESCRIPTION
1	RAS*	Row Address Strobe Output for 16-Pin Dynamic Rams
2	SYSRES*	System Reset Output, Low During Power Up Initialize or Reset Depressed
3	CAS*	Column Address Strobe Output for 16-Pin Dynamic Rams
4	A10	Address Output
5	A12	Address Output
6	A13	Address Output
7	A15	Address Output
8	GND	Signal Ground
9	A11	Address Output
10	A14	Address Output
11	A8	Address Output
12	OUT*	Peripheral Write Strobe Output
13	WR*	Memory Write Strobe Output
14	INTAK*	Interrupt Acknowledge Output
15	RD*	Memory Read Strobe Output
16	MUX	Multiplexor Control Output for 16-Pin Dynamic Rams
17	A9	Address Output
18	D4	Bidirectional Data Bus
19	IN*	Peripheral Read Strobe Output
20	D7	Bidirectional Data Bus
21	INT*	Interrupt Input (Maskable)
22	D1	Bidirectional Data Bus
23	TEST*	A Logic "0" on TEST* Input Tri-States A0-A15, D0-D7, WR*, RD*, IN*, OUT*, RAS*, CAS*, MUX*
24	D6	Bidirectional Data Bus
25	A0	Address Output
26	D3	Bidirectional Data Bus
27	A1	Address Output
28	D5	Bidirectional Data Bus
29	GND	Signal Ground
30	D0	Bidirectional Data Bus
31	A4	Address Bus
32	D2	Bidirectional Data Bus
33	WAIT*	Processor Wait Input, to Allow for Slow Memory
34	A3	Address Output
35	A5	Address Output
36	A7	Address Output
37	GND	Signal Ground
38	A6	Address Output
39	GND	Signal Ground
40	A2	Address Output

NOTE: \*means Negative (Logical "0") True Input or Output



Mates with AMP P/N 88103-1 Card  
Edge Connector or Equivalent

Figure 8-1. TRS-80 edge card connector.

## Adding to the System

In this Chapter we will explore several hardware projects to expand the TRS-80's capabilities :

A parallel printer interface for keyboard units without an expansion interface.

An expansion of system RAM and ROM in a 'reserved' block in the memory map.

Bank selection of RAM and ROM in that 'reserved' blank area, and a memory expansion that includes bank selected RAM.

A programmable input/output port device and a companion interrupt I/O board.

Battery backup and real time clock.

## Parallel Printer Interface

Perhaps the simplest addition to the keyboard unit is a printer interface. Radio Shack sells a complete cable for this purpose, but building one is both less expensive and more enlightening.

When an LLIST or LPRINT command is entered, the computer enters a subroutine which plucks each character to be printed, checks its value, and converts it if necessary. Line feeds, for example, are converted to carriage returns, and form feeds are converted to the proper number of line feeds according to the value in the printer's device control block.

The final value is written to an address, almost exactly as if it were being stored in memory. The hardware decodes that memory

address and sends the character along parallel data lines to the printer. The printer then sends a 'busy' signal to the memory address. The LPRINT or LLIST routines read this busy signal, wasting time until the printer is ready. When the busy signal turns off, the next character is plucked, converted and sent to the printer. The process continues until all characters are printed.

There are disadvantages to this system; among them the fact that the printing process 'hangs up' the computer for the duration. This

can easily be avoided in an interrupt-driven system (see Supplement to Chapter 5), and even modified in a less successful way as a background routine to the keyboard scan. But the overall process is simple, requiring neither sophisticated hardware nor complex software.

Figure 8-1 presents the complete circuit for a printer interface board. It consists of three sections :

1. An address decoder for 37E8, the printer location.

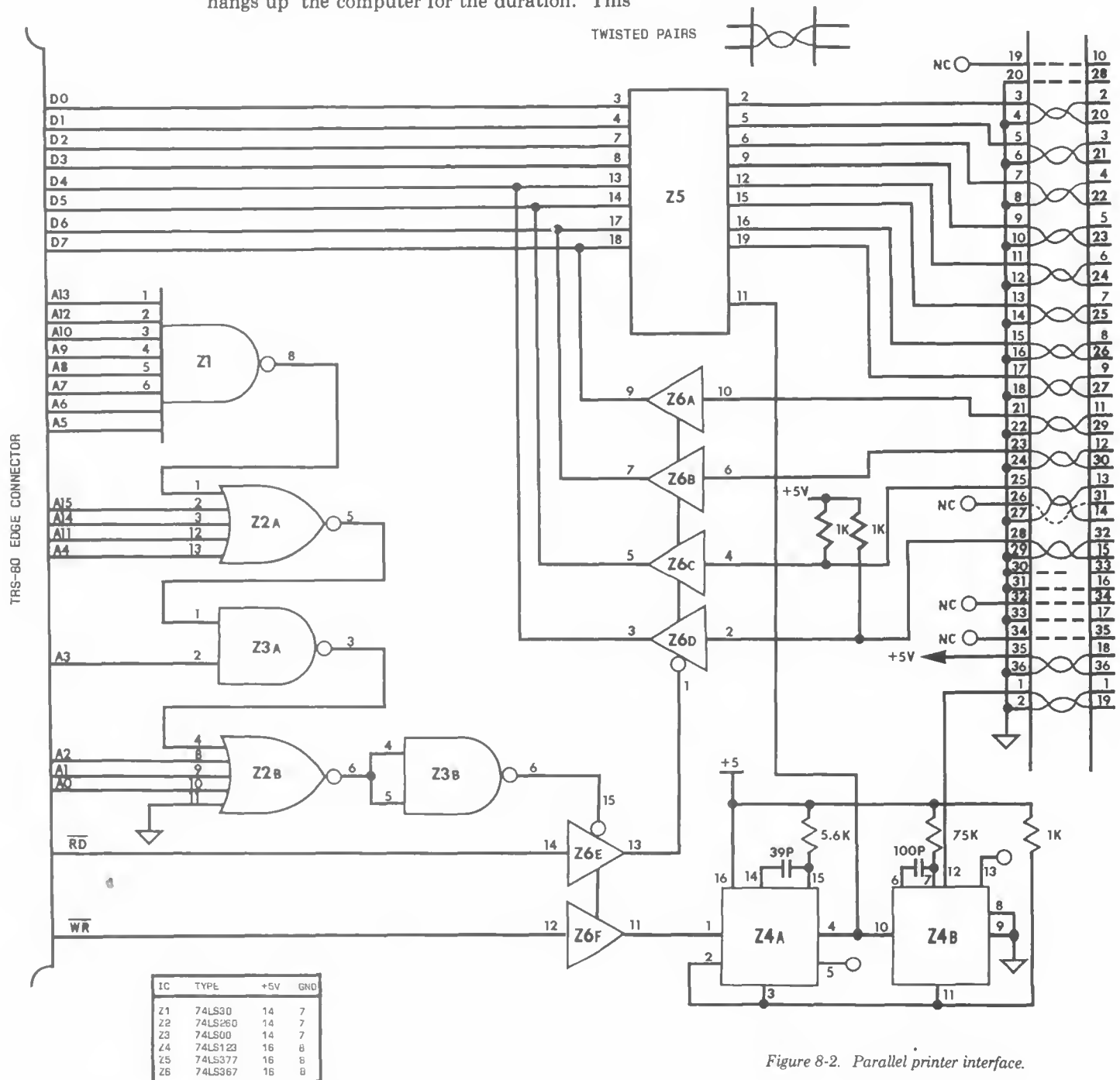


Figure 8-2. Parallel printer interface.

2. An 8-bit output buffer for sending characters to the printer.
3. An input buffer for checking the status of the printer (busy or out of paper).

The address decoder is formed by Z 1 and Z 2 evaluating the address lines in conjunction with RD (read) or WR (write) signals from the computer. If WR is sent, buffer Z 6 is activated, sending a character to the printer. If RD is sent, buffer Z 6 is activated, sending the status of the printer to the CPU.

Power-up the TRS-80, and go through the input sequence. If this process does not act normally, *turn the computer off immediately* and recheck the wiring. If all is normal, it's time for a printer test. Turn the printer on (don't be surprised by a return to MEMORY SIZE? if your printer is an older, electrically noisy one). The simplest way to test the printer interface is to load any program and LLIST it. The printer should spring to life, printing a complete list. If there are problems, they will probably be among the following :

No characters printed at all; computer immediately (or after a short pause) returns to ready.

Characters are being sent to the memory address, but not being received by the printer. Therefore, no 'busy' is being received, and the computer dumps all its characters as fast as it can.

**Solution :** check wiring of the port address, wiring to the output buffer select line, and see that the board has both power and ground wires connected.

No characters printed at all; computer immediately locks up :

Characters are being sent to the memory address, but if none are printed, they are not being received by the printer. The computer is seeing a constant 'busy' signal, and thus is waiting in a loop.

**Solution :** check wiring of the port address, wiring to the input buffer from the printer, and see that the printer is enabled (if it has an enable function).

Intermittent but regular characters (every third or fifth or fiftieth character, for example) are being printed.

Characters are being sent through to the

printer, but no handshake ('busy') is being received by the computer.

**Solution :** check wiring of the input buffer select line, and the wiring from the printer's busy signal. Some printers may not have a busy signal; see box for suggestions..

A single character is printed, then printing stops and the computer locks up.

A constant busy is being received by the computer, and it is waiting in a loop for the busy signal to terminate.

**Solution :** check for ground shorts in the printer's busy line, or shorts to ground at the input buffer.

Incorrect characters are printed, and none or any of the above symptoms are present.

The data lines are incorrectly wired to the printer or to the board's output buffer. The printer may be wired for complement ASCII.

**Solution :** check the wiring of the data lines for reversed wires, either at the computer or printer end. If all is well, enter the following short program :

```
10 FOR X = 65 TO 91
20 Y = NOT X AND 255
30 POKE 14312,Y
40 FOR N = 1 TO 100
50 NEXT : NEXT
```

Listing 8-1. Printer interface test routine.

This program produces the complement of the letters from A to Z. If the correct letters are printed this time, replace Z 5 with an inverting buffer, type 81LS96.

## Talking with the World – The Computer as Boss

The TRS-80 has remarkable skills for controlling the world around it. Four BASIC commands (POKE, PEEK, INP, and OUT) and their machine language equivalents (LD register, LD memory, IN register, OUT port) are the software conversational tools by which the computer makes its wishes known.

Only one dilemma remains: very little hardware was provided with the TRS-80 to use these powerful features. Just a single port (255) was hard-wired in place, and it is limited to controlling cassette functions and video display size. Memory mapped input/output was left exclusively to the expansion interface. And even then then, only for a printer, dual cassette, RS-232 and disks. In each case, no uncommitted user ports or memory addresses were provided.

Fortunately, creating such input/output (I/O) is not difficult. There are two very effective ways to accomplish it:

1. Using inexpensive, separate logic devices that can be dedicated to their interfacing tasks. The TRS-80 Technical Reference Handbook describes such simple hardware in its 'coffee pot' scenario.
2. Using more costly programmable interface devices (such as the INS8255 peripheral interface adaptor) for handling more flexible, general purpose I/O.

In either case, the input/output device must be identifiable by the computer, which means it must somehow be located. It is assigned a port number or a memory address. But what does this number mean, and how does it work? I have often used the analogy of the key in the lock, because it so well describes the way the electronics can open the doors to the world outside its case.

Figure 8-3 shows a simple-minded lock and key. It is simple minded because there are no fine graduations in the height of the tumblers – the 'pins' either rise to a single height, or are not present at all. In the computer, these pins are really voltages, represented by numbers.

In other words, the higher voltage can be called a 'one', and the lower voltage can be considered a 'zero'. In this way, the key number code shown in Figure 8-3 might look like this :

ON	ON	OFF	ON	ON	OFF	OFF	OFF
1	1	0	1	1	0	0	0

The sample key's code (binary 1101 1000) works out to the hexadecimal value D8, or the decimal equivalent 216 (refer to Chapter 2 for details on binary, decimal, and hexadecimal numbers). The 'key' is the value that the computer will output; the 'tumblers' are the hardware which will unlock when this key is inserted.

Below is the schematic of a general purpose 'tumbler' which can be adjusted to open to any electronic key. As noted in Chapter 2, the triangles are buffers which protect the TRS-80 electronic hardware from overexertion. Once again, the triangle with the 'not' circle at its point is an inverting buffer, which reverses the value of any signal placed at its input.

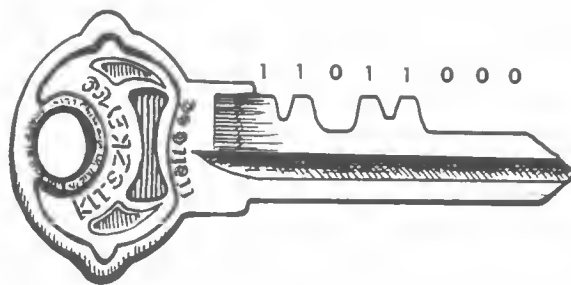


Figure 8-3. Lock-and-key illustration.

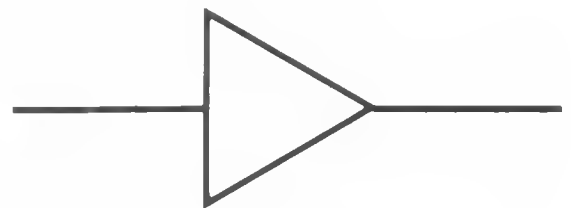


Figure 8-4. Unconnected buffer diagram.

This is very elementary – it will be improved on later – but it is the central scheme of digital operation. Real tumblers must all be lifted to a point so they are in line with the edge of the lock's cylinder. Electronically, the same thing must take place. To turn the electronic cylinder, all the binary input values must be lifted (or depressed) to the same value before the electronic lock will click open.

\*\*\*\*\*;

Using Exclusive-OR for decoding

Resistor 1K ohms to Ground; Switch to Plus Volts		
Switch Position	Input Value	Output Value
OFF (gate sees 0)	0	0
ON (gate sees 1)	0	1
OFF (gate sees 0)	1	1
ON (gate sees 1)	1	0

Resistor 1K ohms to Plus Volts; Switch to Ground		
Switch Position	Input Value	Output Value
OFF (gate sees 1)	0	1
ON (gate sees 0)	0	0
OFF (gate sees 1)	1	0
ON (gate sees 0)	1	1

\*\*\*\*\*;

Table 8-1. Using exclusive-OR for decoding.

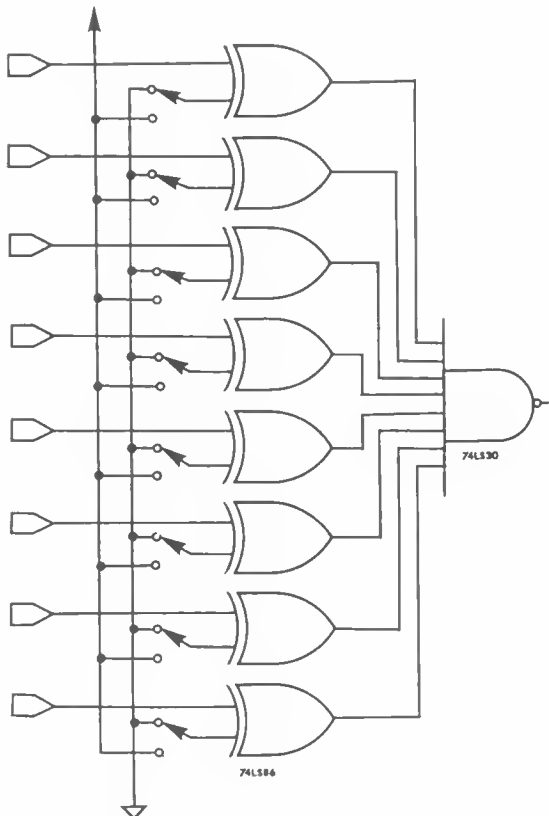


Figure 8-6. Improved port decoding for port addresses.

Figure 8-5 is the schematic for an elementary interface circuit. Z1 and Z2 form the tumblers, and Z3 is the cylinder.

There is another way to produce the same effect as Z1 and Z2, and no rewiring or jumpering is necessary. To do it, you can use one or two 74LS86 exclusive OR gates. Exclusive OR is a remarkable electronic function which states:

*If two input signals are alike, the evaluated result will be set to zero. If two input signals are different, the evaluated output will be set to one.*

Here's how that might work electronically. One input of an exclusive OR gate is attached to a switch and a resistor. The resistor is attached to ground, with the switch connected to the positive voltage line. When the switch is off, the input looks like a zero. When the switch is on, the lower resistance of the switch makes the input of the gate see a one.

Therefore, with the switch off, a signal coming into the second input of the gate would cause a one output when it is a one. With the switch on, a signal coming into the second input of the gate would cause a zero output when it is a one. Refer to Table 8-1.

By using eight exclusive OR gates (two type 74LS86 circuits) and an 8 position DIP (dual inline package) switch, any one of the 256 possible ports can be selected. The circuit below (Figure 8-6) shows how Z1 and Z2 in Figure 8-5 can be replaced with a switch and the 74LS86's to select the port. Thus, jumpering and soldering from the original Z1 and Z2 can be avoided.

Now, before putting these ports to use, it's time to turn to the other type of input/output device – the programmable interface adaptor. The previous I/O device costs perhaps \$5 to create. The central integrated circuit to the programmable I/O port itself costs about \$8, but it offers some extra features and easier wiring.

The INS8255 is a single integrated circuit capable of providing three complete input/output ports. Each port can act as an input or output, and that condition can be changed via programming. This is how it is done: using a decoder similar to that designed above, a 'chip select' is formed.

When the 8255 receives the chip select, it examines its two address line connections. Two address lines can be configured four ways (00, 01, 10, 11), and so can select one of the three I/O

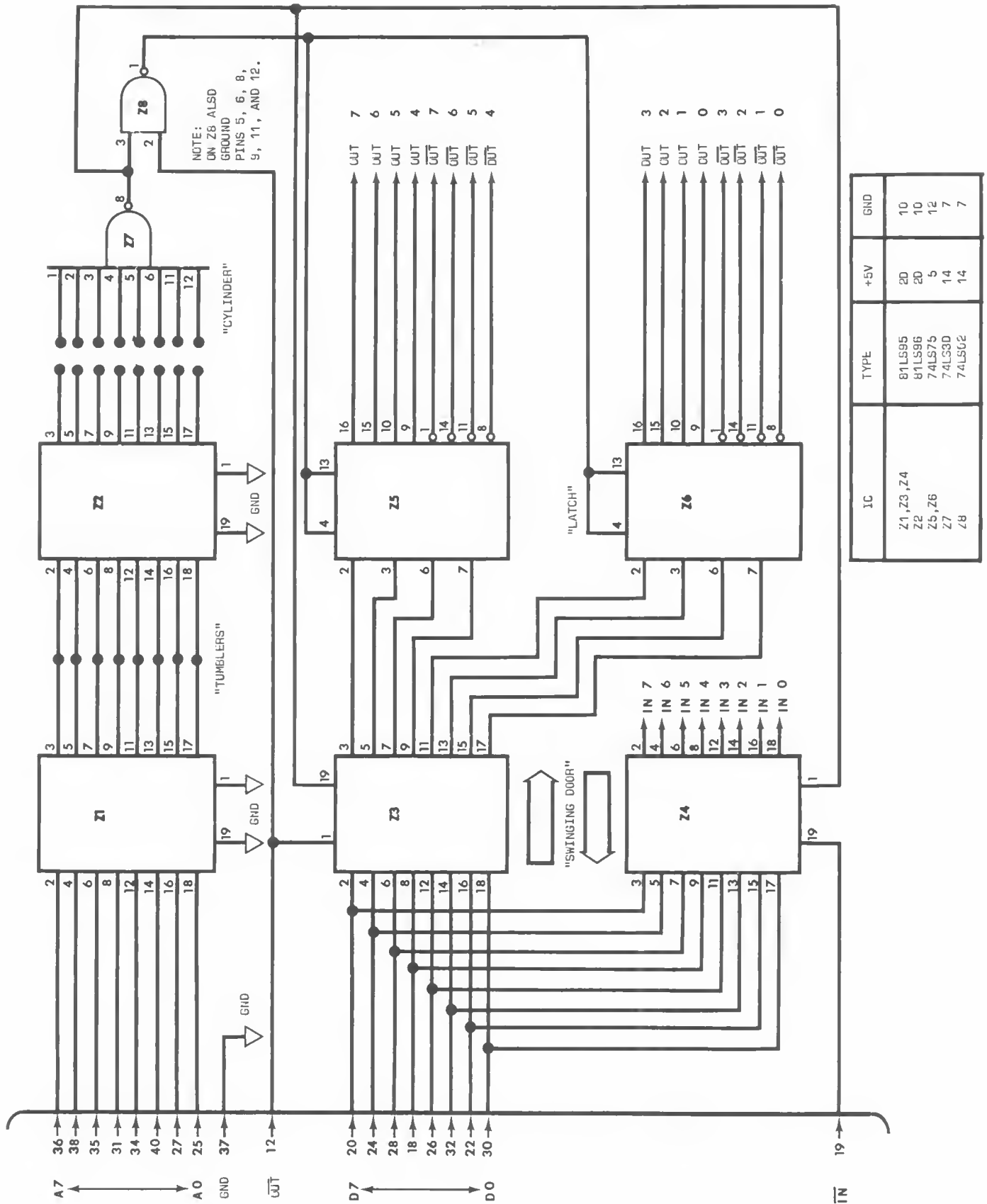


Figure 8-5. Elementary interface.

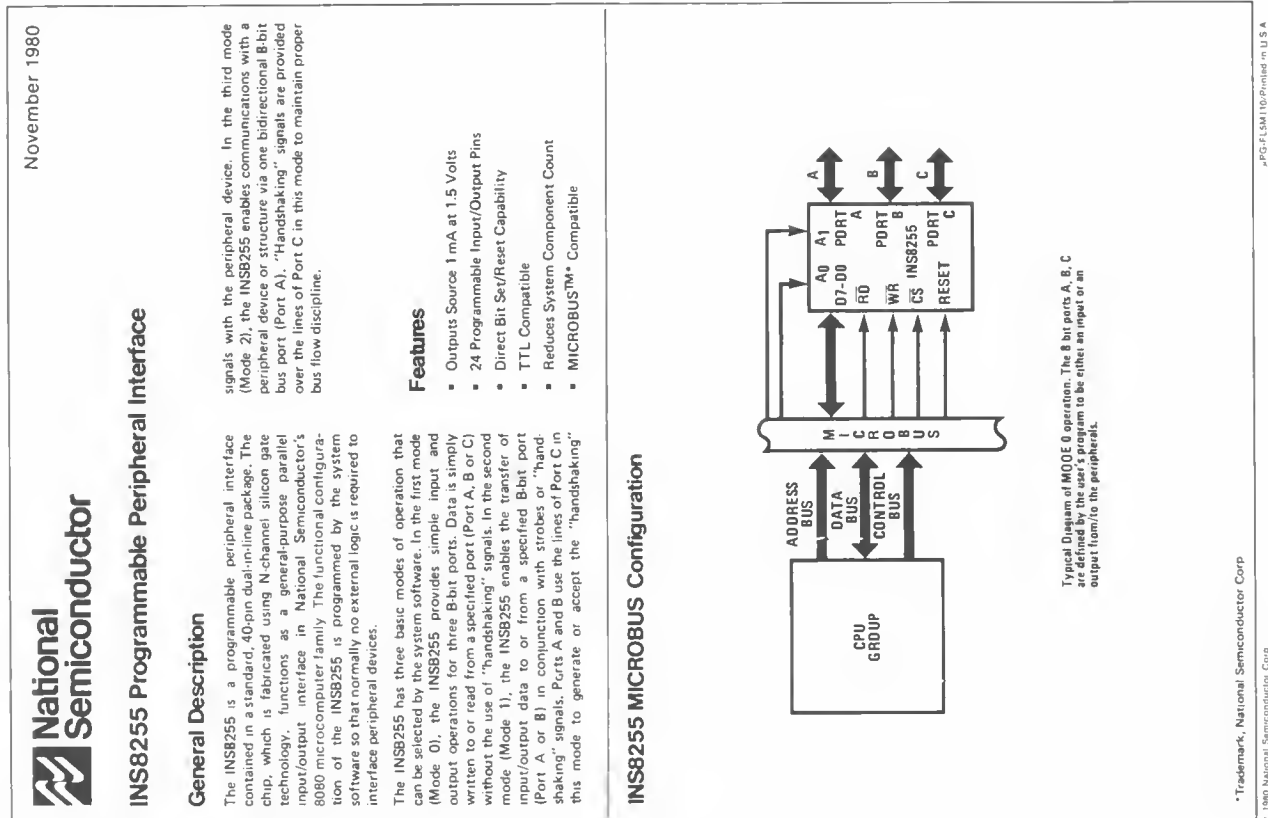


Figure 8-8. 8255 configuration (data).

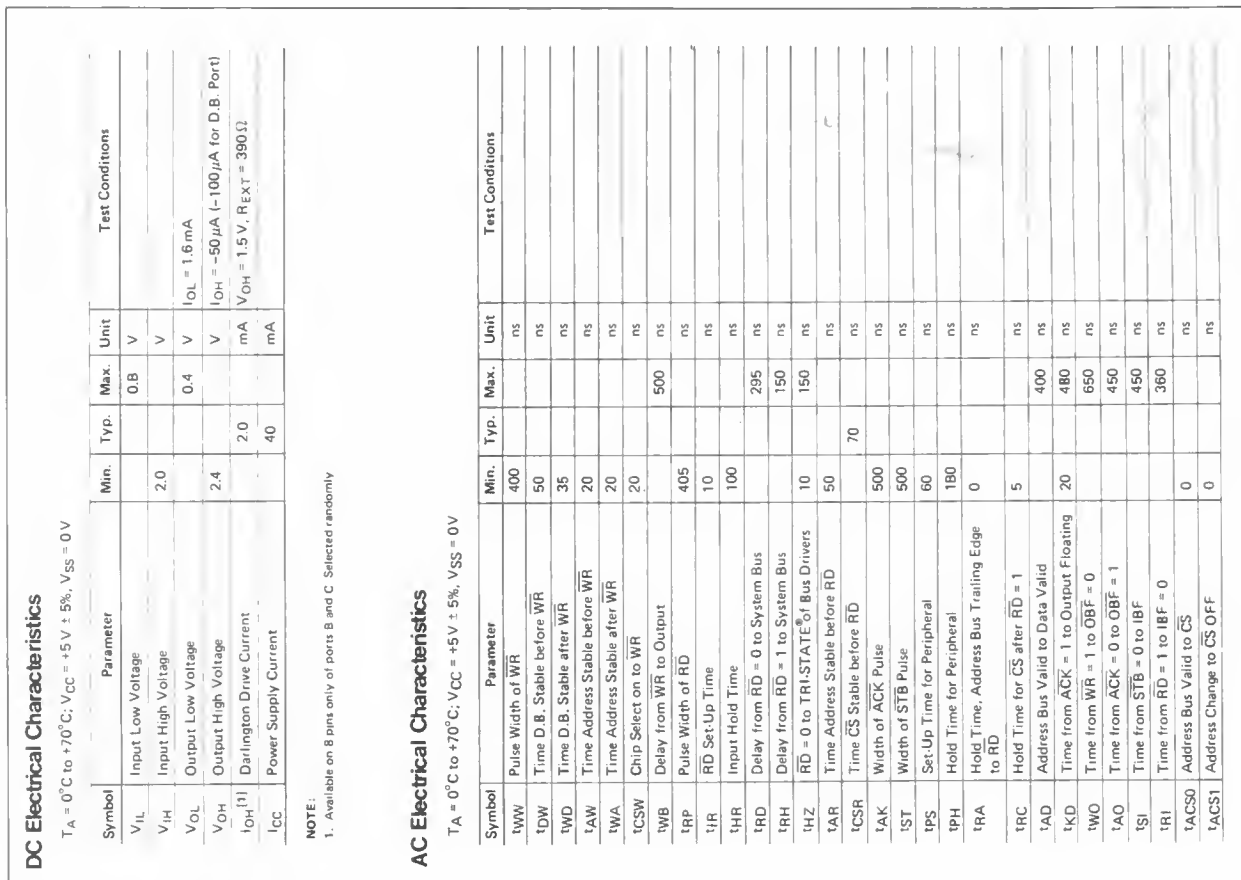


Figure 8-7. 8255 configuration (control).

Figure 8-8. 8255 configuration (data).

ports, or – and this is the remarkable part – the 8255's internal *control register*. The 8255 is a smart chip!

When the control register has been selected, the 8255 can be programmed – its ports may be defined as input or output, and other combinations of actions can be selected. The various possibilities are shown in Figure 8-7 and 8-8.

### Three Real-Time Clock/Calendars

Telling the time and date is a legitimate concern of computer users, not only for keeping documents in order, but also for observing and controlling experiments. Until recently, the only type of real-time clock available was the one built into the expansion interface, which disk and Level III users could access with the TIME\$ command.

The principle of that clock is simple: forty times each second, a pulse is sent from the expansion box to the TRS-80 keyboard unit, where it is applied to the interrupt (INT) line. The computer responds by temporarily setting aside its other processing activities in order to update the seconds, minutes, and hours. When the user asks for the TIME\$, the computer checks the area of memory in which this updated information is stored, and sends it to the current display device.

The clocks presented in this section all use the TIME\$ function, but avoid certain problems associated with the expansion box TIME\$ function. First and foremost, it eliminates the need for the expansion box itself, and the special software to use the 25-millisecond interrupt.

Without disk, this special software must be loaded for every session, reloaded if an inadvertent reset should occur, updated should the computer be turned off, and disabled at every CLOAD and CSAVE. Though convenient in a disk system, this type of approach is more of an annoyance in a Level II system.

There are, however, two distinct advantages to an interrupt-based system for timekeeping: the hardware is simple and cheap to create, and in certain ways the clock pulses generated by the power line are more accurate than crystal-controlled clocks. The power line, because it is linked into a large network of generating systems, must maintain a virtually absolute synchronization over the long term. Short duration lags and leads in the 60-cycle pulses may appear, but the percentage of error over weeks, months, and years is virtually nil.

Time updating from a simple pulse train can be complicated, too, because seconds, minutes and hours, days, weeks, months and years aren't very 'decimal' in their counting. 59 seconds plus one second is one hour and zero seconds. 23 hours and one hour is one day and zero hours. You get the idea. So the software to update a pulse-based calendar needs special math, charts, and tables to keep the time and date in its electronic mind.

HEF. OCT. '80  
80 MICROCOMP  
P. 20

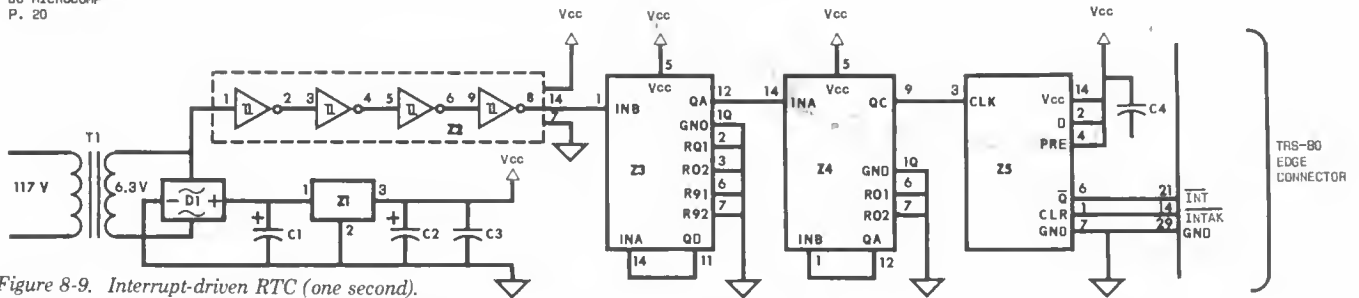


Figure 8-9. Interrupt-driven RTC (one second).

```

00100 ; #####
00110 ; THIS ROUTINE WILL USE THE ONE-SECOND INTERRUPT CREATED
00120 ; BY A HARDWARE ADDITION TO DD TIMEKEEPING. THIS IS
00130 ; TRANSPARENT TO BASIC. NOTE THAT THIS IS NOT INCLUDED
00140 ; IN THE SLASH (/) FUNCTIONS CALLED BY THE CUSTOM
00150 ; INTERPRETER, AND THUS RETURNS ON ITS OWN TO BASIC ROM.
00160 ; #####
00170 ;
7EC0 00180 ORG 7EC0H ; CHANGE TO RELOCATE
00190 ;
00200 ; #####
00210 ; PATCH INTO DOS TIME$ ERROR LOCATION AND CHANGE IT
00220 ; #####
00230 ;
7EC0 F3 00240 ENTRY OI ; DISABLE ACTIVE INTRPTS.
7EC1 210E7E 00250 LO HL,START1 ; ENTRY OF TIME$ PROGRAM
7EC4 227741 00260 LO (4177H),HL ; REPLACE ?L3 ERROR MSG.
7EC7 21A07F 00270 LO HL,START2 ; START OF "CMO" PROGRAM
7ECA 227441 00280 LO (4174H),HL ; REPLACE ?L3 ERROR MSG.
7EC0 3EC3 00290 LO A,0C3H ; GET "JUMP" COMMAND
7ECF 321240 00300 LO (4012H),A ; INSERT INT. PATCH POINT
7E02 214C7F 00310 LO HL,SERVE ; INTERRUPT SERV. ROUTINE
7E05 221340 00320 LO (4013H),HL ; INT. PATCH FROM 003BH
7E08 E056 00330 IM 1 ; SET INTERRUPT MODE #1
7E0A F8 00340 EI ; ENABLE INTERRUPT LINE
7E0B C3CC06 00350 JP 06CCH ; RETURN TO BASIC "READY"
00360 ;
00370 ; #####
00380 ; PATCH TO INTERCEPT ?L3 ERROR AND CHECK LINE'S SYNTAX
00390 ; #####
00400 ;
7E0E 07 00410 START1 RST 10H ; HOUSEKEEP SPACE, ETC.
7E0F E5 00420 PUSH HL ; SAVE BASIC LINE POINTER
7E10 3E11 00430 LO A,11H ; LENGTH OF TIME$ ITSELF
7E12 C05728 00440 CALL 2B57H ; ROM STRING SPACE SETUP
7E15 2A0440 00450 LO HL,(4004H) ; LOCATION TO FIND TIME$
7E18 114340 00460 LO DE,SECOND+2 ; POINT OE TO HOURS POS'N
7E1B C0187F 00470 CALL DISPLY ; CONVERT, PLACE IN TIME$
7E1E 363A 00480 LO (HL),3AH ; PUT COLON INTO TIME$
7E20 23 00490 INC HL ; BUMP TIME$ POINTER
7E21 18 00500 DEC OE ; SET OE TO MINS. POS'N
7E22 C0187F 00510 CALL DISPLY ; CONVERT, PLACE IN TIME$
7E25 363A 00520 LO (HL),3AH ; PUT COLON INTO TIME$
7E27 23 00530 INC HL ; BUMP TIME$ POINTER
7E28 18 00540 DEC OE ; SET OE TO SECS. POS'N
7E29 C0187F 00550 CALL DISPLY ; CONVERT, PLACE IN TIME$
7E2C 3620 00560 LO (HL),20H ; PUT SPACE INTO TIME$
7E2E 23 00570 INC HL ; BUMP TIME$ POINTER
7E2F 114540 00580 LO DE,SECOND+4 ; POINT OE TO MON. POS'N
7E30 C0187F 00590 CALL DISPLY ; CONVERT, PLACE IN TIME$
7E33 362F 00600 LO (HL),2FH ; PUT SLASH INTO TIME$
7E34 23 00610 INC HL ; BUMP TIME$ POINTER
7E35 18 00620 DEC OE ; SET OE TO DAYS POS'N
7E36 C0187F 00630 CALL DISPLY ; CONVERT, PLACE IN TIME$
7E39 362F 00640 LO (HL),2FH ; PUT SLASH INTO TIME$
7E3A 23 00650 INC HL ; BUMP TIME$ POINTER
7E3B 114640 00660 LO DE,SECOND+5 ; POINT OE TO YEARS POS'N
7E3C C0187F 00670 CALL DISPLY ; CONVERT, PLACE IN TIME$
7E3F 3620 00680 JP 2BB4H ; FINISH DISPLAY IN ROM
00690 ;
00700 ; #####
00710 ; FINO VALUES IN TIME LOCATIONS AND CONVERT TO ASCII
00720 ; #####
00730 ;
7F18 1A 00740 DISPLY LO A,(OE) ; GET VALUE INTO ACCUM.
7F19 C0407F 00750 CALL NIBBLE ; SEPARATE INTO 4 BITS
7F1C 47 00760 LO B,A ; VALUE INTO B FOR TEST
7F1D AF 00770 XOR A ; CLEAR A FOR USE IN LOOP
7F1E 04 00780 INC B ; DUMMY INCREMENT ...
7F1F 05 00790 DEC B ; DECREMENT TO TEST FOR 0
7F20 2805 00800 JR Z,LEAVE ; UPPER NIBBLE NOW AT 0
7F22 C616 00810 AOR A,16H ; A=A+16 ...HEX-DEC CONV.
7F24 27 00820 OAA ; DEC.ADJ.: 16 BECOMES 10
7F25 18F8 00830 JR LOOP ; LOOP TILL CONV. DONE
7F27 47 00840 LEAVE LO B,A ; SAVE VALUE BACK IN B
7F28 79 00850 LO A,C ; GET LOW NIBBLE BACK
7F29 FEDA 00860 CP 0AH ; IS IT GREATER THAN 10?
7F2B 3B04 00870 JR C,CLEAN ; NO WORK IF LESS THAN 10

```

Listing 8-2. Interrupt-driven RTC (one second).

Figure 8-9 is a one-pulse-per-second clock that triggers the keyboard unit's interrupt line. Because this clock and the expansion box would compete for that line, this circuit cannot be used in a complete TRS-80 system. But it is a \$5 project, and an inexpensive clock add-on for 16K systems alone.

It consists of five integrated circuits. One regulates the voltage, the second (Z2) takes the sine-wave-shaped signals from the 6.3-volt power transformer and converts them to a sharp-edged digital signal. Z3 and Z4 divide the 60 Hz signal into a one-pulse-per-second signal. Finally, Z5 provides the 1 Hz interrupt signal.

When the interrupting signal is accepted by the TRS-80, an 'interrupt acknowledge' signal is sent back to Z5. There is an important purpose to this action; it turns the interrupt signal off. Why? When the computer receives the interrupt, it sets aside its present software activities to 'service' the interrupt. Unless the interrupting flip-flop is reset, the computer, upon completing the interrupt service routine, will think the previous interrupt is a *new* interrupt, and it will keep updating the time and date.

Software to run this clock is presented in Listing 8-2. It patches into the TIME\$ and CMD locations, and accepts time and date in the following format (use spaces and punctuation exactly as shown):

CMO"09:15:22 05/18/81"

The program checks for correct syntax of the set-time command line, but doesn't verify actual times or dates. So, until the clock is next updated, it will display whatever bizarre time and date you may have set it to!

To print the time and date, enter PRINT TIME\$. You may use TIME\$ just as you would any other strings, including with PRINT, LPRINT, MID\$, LEFT\$, RIGHT\$, concatenation, and other string manipulation. A complete description of these programs is in the supplement to this chapter.

```

7F20 060A 00880 SUB 0AH ; REDUCE IT TO 0 THRU 5
7F2F C610 00880 A00 A,10H ; NOW ADD CARRY BIT
7F31 80 00800 CLEAN A00 A,B ; CREATE A DECIMAL RESULT
7F32 27 00810 OAA ; DEC. ADJ. THE TOTAL
7F33 C0407F 00820 CALL NIBBLE ; SEPARATE INTO 4 BITS
7F36 C630 00830 A00 A,30H ; CONVERT NIBBLE TO ASCII
7F3B 77 00840 LO (HL),A ; PLACE VALUE INTO TIME$
7F39 23 00950 INC HL ; BUMP TIME$ PTR. BY ONE
7F3A 78 00860 LD A,C ; GET VALUE SAVED IN C
7F3B C630 00870 ADD A,30H ; CONVERT NIBBLE TO ASCII
7F30 77 00980 LO (HL),A ; PLACE VALUE INTO TIME$
7F3E 23 00890 INC HL ; BUMP TIME$ PTR. BY ONE
7F3F C8 01000 RET ; BACK TO DO PUNCTUATION
01010 ;
01020 ; #####
01030 ; SUBROUTINE TO CONVERT A BYTE AND SAVE IT AS TWO NIBBLES
01040 ; #####
01050 ;
7F40 F5 01060 NIBBLE PUSH AF ; SAVE THE BYTE BRIEFLY
7F41 E60F 01070 AND OFH ; MASK OUT THE HIGH BITS
7F43 4F 01080 LD C,A ; SAVE LOW NIBBLE IN C
7F44 F1 01080 POP AF ; GET THE WHOLE BYTE BACK
7F45 1F 01100 RRA ; MOVE THE BYTE RIGHT...
7F46 1F 01110 RRA ; ... TWO PLACES ...
7F47 1F 01120 RRA ; ... THREE PLACES ...
7F48 1F 01130 RRA ; UNTIL MSB BECOMES LSB
7F49 E60F 01140 AND OFH ; MASK OUT THE HIGH BITS
7F4B C8 01160 RET ; NIBBLES NOW IN A & C
01180 ;
01170 ; #####
01180 ; INTERRUPT SERVICE ROUTINE IS ENTERED AT 1-S CLOCK PULSE
01180 ; #####
01200 ;
4041 01210 SECOND EQU 4041H ; LOCATION TO STORE TIME$
7F4C F3 01220 SERVE DI ; DON'T BOTHER ME NOW!
7F4D F5 01230 PUSH AF ; SAVE ACCUM. & FLAGS
7F4E E5 01240 PUSH HL ; SAVE HL REGISTER PAIR
7F4F 05 01250 PUSH DE ; SAVE DE REGISTER PAIR
7F50 3A4540 01260 LO A,(SECOND+4) ; GET CURRENT MONTH VALUE
7F53 5F 01270 LO E,A ; SAVE MONTH VALUE IN E
7F54 1600 01280 LO 0,0 ; LET 0=0. REASON FOLLOWS
7F56 214140 01280 LO HL,SECOND ; START AT SECONDS POS'N.
7F58 34 01300 INC (HL) ; SECONDS = SECONDS + 1
7F6A 7E 01310 LO A,(HL) ; GET READY TO COMPARE
7F6B FE3C 01320 CP 600 ; IS IT 60 SECONDS?
7F6D 3B24 01330 JR C,OUT ; DONE IF NOT 60 SECONDS
7F6F C0B87F 01340 CALL TICTOC ; ADVANCE TIME SUBROUTINE
7F82 FE3C 01350 CP 600 ; IS IT 60 MINUTES?
7F84 3B10 01360 JR C,OUT ; DONE IF NOT 60 MINUTES
7F86 C0B87F 01370 CALL TICTOC ; ADVANCE TIME SUBROUTINE
7F89 FE18 01380 CP 240 ; IS IT 24 HOURS?
7F8B 3B16 01380 JR C,OUT ; DONE IF NOT 24 HOURS
7F8D C0B87F 01400 CALL TICTOC ; ADVANCE TIME SUBROUTINE
7F7D E5 01410 PUSH HL ; SAVE REGISTER BRIEFLY
7F71 21B37F 01420 LO HL,LOOKUP ; DAYS-IN-MONTH TABLE
7F74 19 01430 AOD HL,DE ; REMEMBER DE? SEE ABOVE
7F75 8E 01440 CP (HL) ; IS IT LAST DAY OF MONTH
7F76 E1 01450 PDP HL ; GET REGISTER BACK NOW
7F77 3B0A 01460 JR C,OUT ; DONE IF NOT LAST DAY
7F78 C0B87F 01470 CALL TIKTDK ; ADVANCE DATE SUBROUTINE
7F7C FE0D 01480 CP 130 ; IS IT 12 MONTHS?
7F7E 3B03 01490 JR C,OUT ; DONE IF NOT 12 MONTHS
7F8D C0B87F 01500 CALL TIKTOK ; ADVANCE DATE SUBROUTINE
7F83 01 01510 OUT POP DE ; RESTORE DE REGISTERS
7F84 E1 01520 POP HL ; RESTORE HL REGISTERS
7F85 F1 01530 PDP AF ; RESTORE ACCUM. & FLAGS
7F86 FB 01540 EI ; GET CLCK TICKING AGAIN
7F87 ED40 01550 RETI ; BACK FROM THE INTERRUPT
01560 ;
01570 ; #####
01580 ; ADVANCE TIME/DATE & RETRIEVE NEW VALUE SUBROUTINES
01590 ; #####
01600 ;
7F8B AF 01610 TICTOC XDR A ; CLEAR ACCUM. TO ZERO
7F8A 77 01620 FINISH LD (HL),A ; HRS, MIN, OR SEC = 0
7F8B 23 01630 INC HL ; MOVE TO NEXT POSITION
7F8C 34 01640 INC (HL) ; TIME = TIME + 1 (CARRY)
7F8D 7E 01650 LD A,(HL) ; SET UP TO TEST VALUE
7F8E C8 01660 RET ; BACK TO COMPLETE TEST
7F8F C601 01670 TIKTDK LD A,1 ; A = 1 FOR DAY OR MONTH
7F91 1B7F 01680 JR FINISH ; OTHER ROUTINE DOES WORK
01680 ;
01700 ; #####
01710 ; THIS IS THE DAYS-IN-A-MONTH LOOKUP TABLE - NO LEAP YEAR
01720 ; #####
01730 ;
7F93 00 01740 LOOKUP 0EF8 00 ; DUMMY BYTE, BUT THEN...
7F94 20 01750 0EF8 320 ; THIRTY DAYS HATH
7F95 1D 01760 0EF8 28D ; SEPTEMBER,
7F96 20 01770 0EF8 320 ; APRIL, JUNE, AND
7F97 1F 01780 0EF8 31D ; NOVEMBER;
7F98 20 01790 0EF8 320 ; ALL THE REST HAVE
7F99 1F 01800 0EF8 310 ; THIRTY-ONE,
7F9A 20 01810 0EF8 320 ; 'CEPT FEBRUARY, AND
7F9B 20 01820 0EF8 320 ; THERE'S BEEN ALL
7F9C 1F 01830 0EF8 31D ; TOO MUCH TALK
7F9D 20 01840 0EF8 32D ; ABOUT THE MYRIAD
7F9E 1F 01850 0EF8 310 ; PRETIOIGITATIONS
7F9F 20 01860 0EF8 32D ; USING THAT MONTH
01870 ;
01880 ; #####
01890 ; "CMD" PATCH CHECKS PARAMETERS, SYNTAX, AND SETS TIME
01900 ; #####

```

The second and third clocks for the TRS-80 are similar in concept, but different in execution. That difference has only to do with manufacturing quality of the specified clock-calendar chip, the MSM5832 (available from *Digi-Key Corporation* and *Hobbyworld Electronics* — see Appendix). This clock-calendar has been designed to interface with microcomputers instead of the familiar red LED readouts. A 32.768 KHz crystal is required for its operation (also sold by the above suppliers).

It provides time in hours (12 or 24), minutes and seconds; month, day, year (leap year as well) and day of the week. It has timing signal outputs for interrupt use, which will not be used in this circuit. A battery backup will keep it in time when the TRS-80 is turned off. Two complete circuits are presented in Figure 8-10 and Figure 8-11

Why two circuits? Because the MSM5832 is a relatively slow electronic circuit, and, depending on the quality of the production run used for the chip you purchase, it may or may not be fast enough to interface directly with your TRS-80!

For a slower chip, you will need to use intermediate logic to latch onto the clock information in its own good time, and feed it to the TRS-80 as the computer's signals speed past. For this job, the INS8255 (as recommended by *OKI*, manufacturers of the MSM5832) is used. As noted above, the 8255 is a peripheral interface adapter, which sets up a private, latched bus between itself and the clock chip. Clock information is sent via Port A. The clock chip's address lines are selected through Port B, and other timekeeping features are selected through Port C.

The 8255 will be placed in the TRS-80 memory map at 37D0 through 37D2, below the cassette/ disk latches, and above the Level II operating system. Z1 and Z2 in Figure 8-10 decode that address group, and pins 8 and 9 of the 8255 (Z3) are used to select the specific address among those.

The last circuit, Figure 8-11, places the MSM5832 directly into the TRS-80 memory map without use of the 8255 interface chip. Because the MSM5832 has four address lines, it is decoded differently from the 8255, but occupies the same general area (37D0 to 37DF).

Wiring all these clocks is a simple procedure because there are few parts. All can be soldered or wire wrapped, though sockets are virtually essential for the 8255 and MSM5832 chips. The latter is a static-sensitive chip, and should be



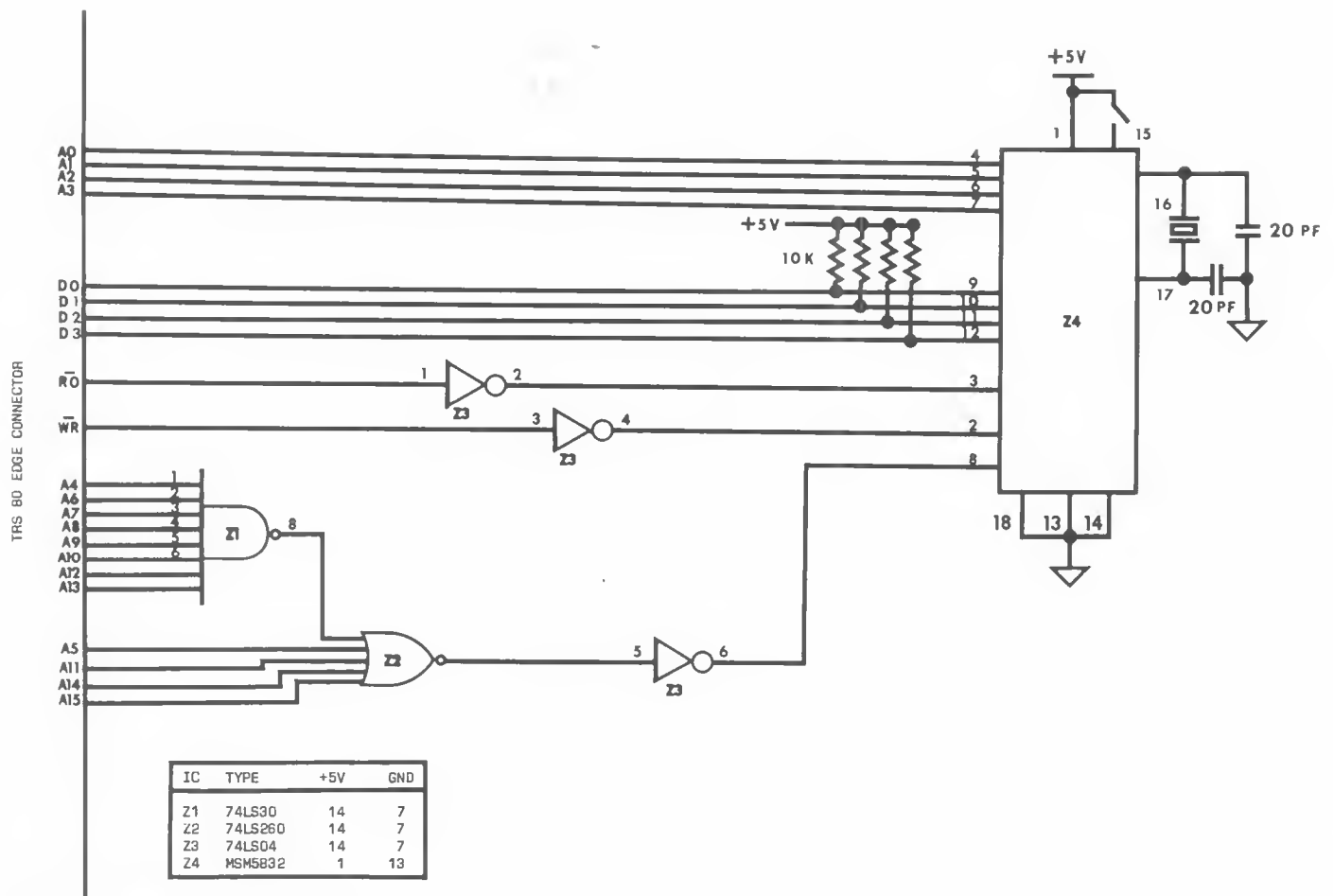
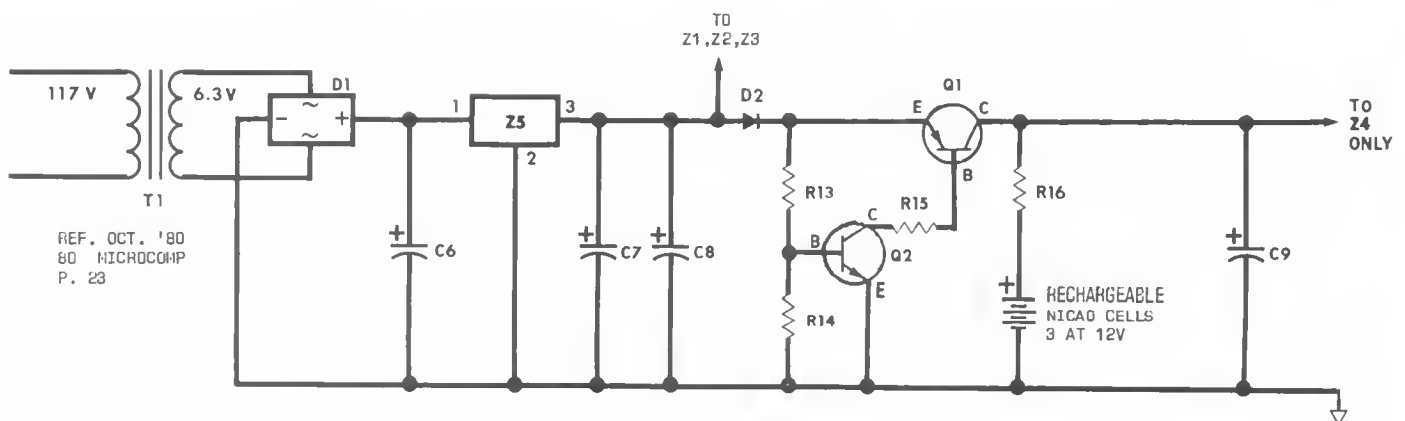


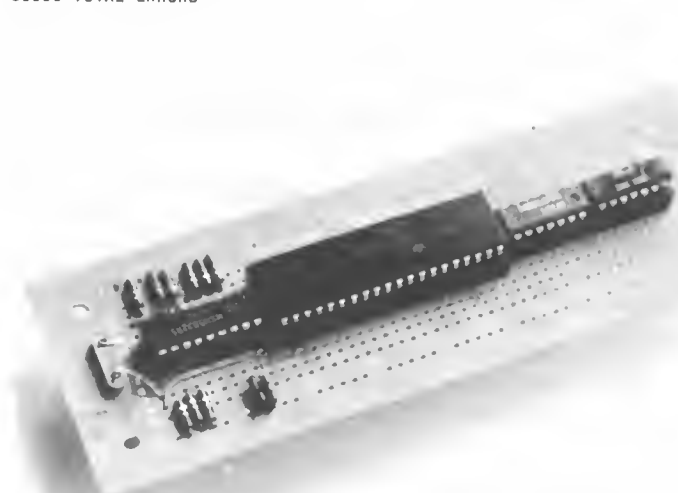
Figure 8-11. MSM5832 direct to bus RTC.



```

01910 ;
7FA0 114340 01920 START2 LO 0E,SECOND+2 ; POINT 0E TO HOURS POS'N
7FA3 7E 01930 LO A,(HL) ; CHAR AT LINE POINTER
7FA4 FE22 01940 CP 22H ; IS IT A QUOTE MARK?
7FA8 204A 01950 JR NZ,OTHERS ; CHECK FOR CMOT OR CMOR
7FAB C0087F 01960 CALL CONVRT ; READ/CONV. ASCII HR.
7FAB FE3A 01970 CP 3AH ; IS IT A COLON?
7FA0 C2971B 01980 SYNERR JP NZ,1987H ; GO TO ?SN ERROR ROUTINE
7F90 C0087F 01990 CALL CONVRT ; READ/CONV. ASCII MIN.
7F93 FE3A 02000 CP 3AH ; IS IT A COLON?
7F95 20F6 02010 JR NZ,SYNERR ; SYNTAX ERROR IF NOT :
7F97 C0087F 02020 CALL CONVRT ; READ/CONV. ASCII SEC.
7F9A FE20 02030 CP 20H ; IS IT A SPACE?
7F9C 20EF 02040 JR NZ,SYNERR ; SYNTAX ERROR IF NOT
7FBE 114840 02050 LO 0E,SECOND+4 ; POINT 0E TO MONTH POS'N
7FC1 C0087F 02060 CALL CONVRT ; READ/CONV. ASCII MON.
7FC4 FE2F 02070 CP 2FH ; IS IT A SLASH?
7FC6 20E5 02080 JR NZ,SYNERR ; SYNTAX ERROR IF NOT /
7FC8 C0087F 02090 CALL CONVRT ; READ/CONV. ASCII DAY
7FC9 FE2F 02100 CP 2FH ; IS IT A SLASH?
7FC0 200E 02110 JR NZ,SYNERR ; SYNTAX ERROR IF NOT /
7FCF 114640 02120 LO 0E,SECOND+S ; POINT 0E TO YEARS POS'N
7F02 C0087F 02130 CALL CONVRT ; READ/CONV. ASCII YEAR
7F05 FE22 02140 CP 22H ; IS IT A QUOTE MARK?
7F07 2001 02150 JR NZ,EXIT ; DONE IF A QUOTE MARK
7F09 23 02160 INC ML ; BUMP POINTER PAST QUOTE
7F0A CB 02170 EXIT RET ; BACK TO BASIC
02180 ;
02190 ; #####
02200 ; CONVERT ASCII TO HEX AND POKE INTO CLOCK TIME$ LOCATION
02210 ; #####
02220 ;
02230 CONVRT INC HL ; BUMP LINE PTR. BY ONE
02240 LO A,(HL) ; GET CHARACTER IN LINE
02250 SUB 30H ; CONVERT ASCII TO HEX
02260 INC A ; MAKE A 9E AT LEAST 1
02270 LO 9,A ; SAVE THAT VALUE IN B
02280 LO A,0F6H ; A= 100 HEX MINUS 10 DEC
02290 MULT ADO ; MULTIPLY 8Y ADDITION
02300 DJNZ MULT ; I.E., A = 9 TIMES 10
02310 LO 9,A ; SAVE THAT VALUE IN B
02320 INC HL ; BUMP LINE PTR. 8Y ONE
02330 LO A,(ML) ; GET CHARACTER IN LINE
02340 SUB 30H ; CONVERT ASCII TO HEX
02350 ADO A,B ; A = (9 * 10) + A
02360 LO (0E),A ; TIME IS SET, PUT IN 0E
02370 DEC 0E ; BUMP 0E TO NEXT PLACE
02380 INC ML ; BUMP LINE PTR. 8Y ONE
02390 LO A,(HL) ; GET CHARACTER IN LINE
02400 RET ; RETURN FOR FURTHER TEST
02410 OTHERS CP 52H ; IS IT CMOR (CLOCK OFF)?
02420 JR NZ,NEXT ; NOPE, TRY FOR CMOT
02430 OI ; TURN OFF THE CLOCK
02440 INC ML ; BUMP LINE PTR. 8Y ONE
02450 RET ; BACK TO BASIC PROGRAM
02460 NEXT CP 54H ; IS IT CMOT (CLOCK ON)?
02470 JR NZ,SYNERR ; NOPE, MUST BE ERROR
02480 EI ; TURN ON THE CLOCK
02490 INC ML ; BUMP LINE PTR. 8Y ONE
02500 RET ; BACK TO BASIC PROGRAM
02510 ;
02520 ; #####
02530 END ENTRY
7EC0 00000 TOTAL ERRORS

```



MSM5832 clock board contains only 4 integrated circuits, some resistors, and a crystal. Power supply is external, and includes rechargeable batteries for backup.

handled carefully (see Chapter 4 for details). Use wires as short as possible in the area where the crystal is located, and triple-check all connections before applying the power. The MSM5832 is a delicate circuit, and at \$9 a shot, worth the trouble to check your work. The circuits are connected to the computer via standard edge-card connectors.

### Edge Card Connectors: What's Up?

The 40 pin bus of the TRS-80 is a non-standard animal to begin with, and is not particularly logical in its pin assignments. What makes things more frustrating than merely locating the position of the pins, is trying to figure out which way is up.

The edge card of the TRS-80, when viewed from its edge, has pin 1 located at the top left – a logical place. Naturally, because the CPU and expansion unit face each other, the entry to the expansion interface is the mirror image of this design. Yet its top left pin is also called 'pin 1'! So pin 1 (TRS-80) leads to pin 39 (expansion box).

It gets stickier than this, however. When you are building projects, it's likely you won't be doing it with professionally etched edge cards, but rather with headers and wires and cables of various kinds which you assemble yourself. So when you get your 40 pin edge connector to hook onto the TRS-80, notice –

- the numbering of the pins on the connector, which, if industry standards are used, has pin 1 marked on the opposite side, like the expansion box.
- the outside wire, which may either lead to pin 1 or pin 2, depending on the manufacturer.
- the orientation of the connecting header, which may reverse the process one more time.

To be absolutely sure, use a meter to determine the path all the way from the computer edge card to the final connector when it is mounted on the project. It's best to mount the header or connector first, identify its pins with a meter, mark them and *then* begin work wire wrapping or soldering.

```

10 CLS : CLEAR 150 : REM * CRUDE BUT SERVICEABLE CLOCK PROGRAM
20 FOR X = 0 TO 6 : READ OW$(X) : NEXT : REM * ARRAY OF DAYS
30 DATA M O N O A Y, T U E S D A Y, W E D N E S D A Y
40 DATA T H U R S D A Y, F R I D A Y, S A T U R D A Y, S U N D A Y
50 PRINT "ENTER HOURS AND MINUTES, PLUS AM OR PM INDICATION."
60 INPUT "USE FORMAT 0,3,5,8,P (= 3:58 P.M.)":H0,H1,M0,M1,P$
70 INPUT "12-HOUR OR 24-HOUR CLOCK" (ANSWER 12 OR 24):C$
80 IF P$ = "P" THEN H0 = H0 + 4 : REM * 8IT 3 INDICATES P.M.
90 IF C$ = "24" THEN H0 = H0 + 8 : REM * 8IT 4 FOR 24 HOURS
100 PRINT "DAY OF THE WEEK (ENTER 1 TO 7. MONDAY IS 1.)"
110 INPUT OW : OW = OW - 1 : REM * CLOCK'S MONDAY IS ZERO
120 PRINT "MONTH, DAY AND YEAR IN FORMAT 0,3,3,1,8,0 (3/31/80)"
130 INPUT M2,M3,00,01,Y0,Y1 : REM * LEAP YEAR TEST IN NEXT LINE
140 LY = Y0 + 10 * Y1 : IF LY/4 = FIX (LY/4) THEN O0 = O0 + 4
150 POKE 14291,128 : REM * SET UP 8255 CHIP PORTS
160 POKE 14290,80 : REM * SET UP CLOCK TO READ TIME AND DATE
170 Q = 14289 : REM * THIS IS CLOCK ADDRESS REGISTER
180 POKE Q,2 : POKE Q-1,M1 : POKE Q,3 : POKE Q-1,M0
190 POKE Q,4 : POKE Q-1,H1 : POKE Q,5 : POKE Q-1,H0
200 POKE Q,6 : POKE Q-1,O0 : POKE Q,7 : POKE Q-1,O1
210 POKE Q,8 : POKE Q-1,O0 : POKE Q,9 : POKE Q-1,M3
220 POKE Q,10 : POKE Q-1,M2 : POKE Q,11 : POKE Q-1,Y1
230 POKE Q,12 : POKE Q-1,Y0 : REM * TIME AND DATE INFO SET
240 POKE 14291,144 : CLS : REM * DISPLAY SUBROUTINE FOLLOWS
250 PRINT @ 0, "": : REM * DISPLAY IS ON TOP LINE OF SCREEN
260 POKE 14280,32 : REM * SET UP CLOCK TO WRITE TIME AND DATE
270 POKE Q,6 : PRINT OW$ (PEEK [Q-1] AND 1S), "":
280 POKE Q,10 : PRINT PEEK [Q-1] AND 15; CHR$(8);
290 POKE Q,9 : PRINT PEEK [Q-1] AND 15; "/";
300 POKE Q,8 : PRINT PEEK [Q-1] AND 3; CHR$(8);
310 POKE Q,7 : PRINT PEEK [Q-1] AND 15; "/";
320 POKE Q,12 : PRINT PEEK [Q-1] AND 15; CHR$(8);
330 POKE Q,11 : PRINT PEEK [Q-1] AND 15; " -- ";
340 POKE Q,5 : PRINT PEEK [Q-1] AND 3; CHR$(8);
350 POKE Q,4 : PRINT PEEK [Q-1] AND 15; " : ";
360 POKE Q,3 : PRINT PEEK [Q-1] AND 15; CHR$(8);
370 POKE Q,2 : PRINT PEEK [Q-1] AND 15; " : ";
380 POKE Q,1 : PRINT PEEK [Q-1] AND 15; CHR$(8);
390 POKE Q,0 : PRINT PEEK [Q-1] AND 15;
400 POKE Q,5 : IF (PEEK [Q-1] AND 4) = 0 THEN PRINT " A. M.";
410 : IF (PEEK [Q-1] AND 4) = 4 THEN PRINT " P. M.";
420 IF (PEEK[143121 AND 128] = 0 THEN 500 ELSE 250
430 REM : THIS ROUTINE IS MUCH LONGER THAN IT NEED BE,
440 REM : BUT IS SET UP FOR CLARITY, NOT EFFICIENT USE
450 REM : OF MEMORY. IT IS EASIER TO USE THE MACHINE
460 REM : LANGUAGE SUBROUTINE FOR THIS CLOCK CIRCUIT.
500 A$ = "": FOR X = 15360 TO 15424 : A$ = A$ + CHR$(PEEK(X) )
510 NEXT : LPRINT A$ : GOTO 250

```

Listing 8-3. BASIC program for MSM5832/8255.

```

00100 : #####
00110 : MACHINE LANGUAGE CLOCK PROGRAM FOR MSM5832 CLOCK BOARD
00120 : ATTACHED TO THE TRS-80 AT ADDRESS 142B8 TO 14300, USING
00130 : THE "TIME$" FUNCTION. THIS ROUTINE IS TRANSPARENT TO
00140 : BUT IS NOT CALLED BY THE CUSTOM INTERPRETER. HENCE,
00150 : IT OPERATES INDEPENDENTLY FROM THE INTERPRETER PATCH.
00160 : #####
00170 :
7E00 00180 ORG 7E00H ; CHANGE TO RELOCATE
00190 :
00200 : #####
00210 : PATCH INTO DOS TIME$ ERROR LOCATION AND CHANGE IT
00220 : #####
00230 :
7E00 210F7E 00240 ENTRY LD HL,START1 ; START OF TIME$ PROGRAM
7E03 227741 00250 LD (4177H),HL ; PATCH TIME$ ?L3 ERROR
7E06 218F7E 00260 LD HL,START2 ; START OF "CMD" PROGRAM
7E09 227441 00270 LD (4174H),HL ; PATCH CMD ?L3 ERROR
7E0C C3CC06 00280 JP 06CCH ; BACK TO A BASIC "READY"
00290 :
00300 : #####
00310 : THIS IS THE BEGINNING OF TH "TIME$" PATCH TO READ TIME.
00320 : ROUTINE INTERCEPTS ?L3 ERROR AND CHECKS LINE'S SYNTAX.
00330 : #####
00340 :
7E0F 07 00350 START1 RST 10H ; BASIC HOUSEKEEPING
7E10 ES 00360 PUSH HL ; SAVE BASIC LINE POINTER
7E11 3E18 00370 LD A,18H ; LENGTH OF TIME$
7E13 C05728 00380 CALL 28S7H ; ROM STRING SPACE SETUP
00390 :
00400 : #####
00410 : SET UP RAM SPACE AND GET CLOCK CHIP READY TO READ TIME
00420 : #####
00430 :
7E16 2A0440 00440 LD HL,(4004H) ; LOCATION TO STORE TIME$
7E19 F0210037 00450 LD IY,3700H ; CLOCK MEMORY ADDRESS
7E10 F0360390 00460 LD (IY+3),90H ; SET UP 8255 CHIP PORTS
7E21 C06A7F 00470 CALL DELAY ; WAIT FOR SLOW MSM5832
7E24 F0360220 00480 LD (IY+2),20H ; SET UP CLOCK TO READ
7E28 C06A7F 00490 CALL DELAY ; WAIT FOR SLOW MSM5832
00500 :
00510 : #####
00520 : CLOCK IS READY TO READ ... NOW READ AND CREATE STRING.
00530 : DAY OF THE WEEK IS ALPHABETIC AND WILL BE DONE FIRST.
00540 : #####
00550 :

```

Listing 8-4. Assembly program for MSM5832/8255.

To use the MSM5832 clock with the 8255 interface adaptor, you will need to refer to both chips' programming information. Figure 8-(?), earlier in this chapter, contains the 8255 programming parameters. Figure 8-(?) shows how the clock's registers are set up.

At first, this process may appear confusing. What is being done? Three semi-intelligent electronic devices are being taught to talk to one another. The TRS-80 knows it wants to read and write to memory. The 8255 is that memory. But the 8255 has a mind of its own, and that mind can only be controlled by selecting its control register, telling it what purpose each of its three ports is to serve, and then reading and writing those ports. Finally, the MSM5832 also has a mind of its own. It will neither report nor accept the time until it is told what aspect of the time is needed, and that too is done via a control register.

Figure 8-(?) is a flow chart which describes the process, and Listing 8-(?) is a BASIC program which fairly well describes the steps needed to access the MSM5832 chip, 'way down the chain.

Eliminating the 8255 by using the second circuit means only one set of electronic parts must be taught to speak to each other. The TRS-80 can probe right into the MSM5832 control register to select which aspect of the time and date it wishes.

Each of the machine language programs presented in Listings 8-(?) and 8-(?) use the TIME\$ and CMD commands to set and recall the time. To set the time and date, enter :

CMD"MON 03/14/49 02:29 PM"

Notice that this differs from the interrupt driven clock (Listing 8-(?)) in that the day of the week and morning - afternoon indicators must be given. Remember to use the punctuation and spacing exactly as printed here. As with the interrupt clock, PRINT TIME\$ returns the time and date, and this TIME\$ can be used and manipulated just as any other string.

A complete description of these programs is given in the Supplement to this Chapter.

```

7E2B F0360106 00560      LD      [IY+1],6      ; POINT TO DAY OF WEEK
7E2F C06A7F 00570      CALL    DELAY          ; WAIT FOR SLOW MSM5832
7E32 F07E00 00580      LD      A,[IY+0]      ; GET DUMMY VALUE INTO A
7E35 C06A7F 00590      CALL    DELAY          ; WAIT FOR SLOW CHIP (1)
7E38 F07E00 00600      LD      A,[IY+0]      ; GET DAY OF WEEK VALUE
7E3B E607 00610      .AND    07H          ; MASK OFF UNUSED BITS
7E3D 11757F 00620      LD      0E,TABLE      ; POINT OE TO DAY TABLE
7E40 3C 00630      INC      A          ; IT MUST BE AT LEAST 1
7E41 30 00640      L00P1  DEC      A          ; IS ACCUMULATOR ZERO?
7E42 2807 00650      JR      Z,XLOOP      ; GO OUT OF TABLE LOOP
7E44 0603 00660      LD      8,3          ; NUMBER OF CHARS PER DAY
7E46 13 00670      L00P2  INC      0E          ; MOVE PAST EACH CHAR
7E47 10FD 00680      DJNZ    L00P2      ; DO IT TILL AT NEXT DAY
7E49 1BF6 00690      JR      L00P1      ; CHECK FOR NEXT DAY

00700 ;
00710 ; #####
00720 ; VALUE FOR DAY IS FOUND ... NOW TURN IT INTO LETTERS
00730 ; #####
00740 ;

7E4B 0603 00750      XLDDP   LD      B,3          ; NUMBER OF CHARS TO GET
7E4D 1A 00760      YLDDP   LD      A,[0E]      ; CHARACTER TO TRANSFER
7E4E 77 00770      LD      [HL],A          ; XFER DAY NAME TO TIME$
7E4F 23 00780      INC      HL          ; NEXT LOCATION IN TIME$
7E50 13 00790      INC      0E          ; NEXT LOCATION IN TABLE
7E51 10FA 00800      DJNZ    YLDDP      ; LOOP BACK FOR NEXT CHAR
7E53 362D 00810      LD      [HL],20H      ; PUT SPACE AFTER DAY
7E55 23 00820      INC      HL          ; BUMP TIME BUFFER AGAIN
00830 ;
00840 ; #####
00850 ; DAY OF WEEK IS DONE ... NOW GET MONTH, DAY, AND YEAR
00860 ; #####
00870 ;

7E56 1E30 00880      LD      E,30H          ; HEX TO ASCII DIFFERENCE
7E58 1608 00890      LD      0,11          ; MONTH HI PORT + 1
7E5A 062F 00900      LD      B,2FH          ; SLASH ["/"] CHARACTER
7E5C 0E0F 00910      LD      C,DFH          ; MASK UNUSED PORT BITS
7E5E C0557F 00920      CALL    FILLER          ; GET MONTH HIGH VALUE
7E61 C0557F 00930      CALL    FILLER          ; GET MONTH LOW VALUE
7E64 70 00940      LD      [HL],B          ; LOAD SLASH INTO TIME$
7E65 23 00950      INC      HL          ; BUMP TIME BUFFER BY ONE
7E66 0E03 00960      LD      C,3          ; MASK UNUSED CLOCK BITS
7E68 C0557F 00970      CALL    FILLER          ; GET DAY HIGH VALUE
7E6B 0E0F 00980      LD      C,DFH          ; MASK UNUSED CLOCK BITS
7E6D C0557F 00990      CALL    FILLER          ; GET DAY LOW VALUE
7E70 70 01000      LD      [HL],B          ; PUT SLASH INTO TIME$
7E71 23 01010      INC      HL          ; BUMP TIME BUFFER BY ONE
7E72 160D 01020      LD      D,13          ; YEAR HIGH VALUE + 1
7E74 C0557F 01030      CALL    FILLER          ; GET YEAR HIGH VALUE
7E77 C0557F 01040      CALL    FILLER          ; GET YEAR LOW VALUE
7E7A 362D 01050      LD      [HL],20H      ; VALUE FOR A SPACE
7E7C 23 01060      INC      HL          ; BUMP TIME BUFFER BY ONE
01070 ;
01080 ; #####
01090 ; MONTH, DAY, YEAR DONE - NOW GET HOURS, MINUTES, SECONDS
01100 ; #####
01110 ;

7E7D 1605 01120      LD      0,5          ; HOURS HIGH VALUE
7E7F F07201 01130      LD      [IY+1],0      ; SET UP CLOCK CHIP PORT
7E82 C06A7F 01140      CALL    DELAY          ; DELAY FOR 8255 CHIP
7E85 F07E00 01150      LD      A,[IY+0]      ; DUMMY VALUE INTO ACC.
7E88 C06A7F 01160      CALL    DELAY          ; DELAY AGAIN FOR CHIP1
7E8B F07E00 01170      LD      A,[IY+0]      ; GET HOURS HIGH VALUE
7E8E F5 01180      PUSH    AF          ; SAVE THIS FOR AM/PM
7E8F 14 01190      INC      0          ; ACCOMMODATE SUBROUTINE
7E90 0E03 01200      LD      C,3          ; MASK UNUSED CLOCK BITS
7E92 C0557F 01210      CALL    FILLER          ; GET HOURS HIGH VALUE
7E95 0E0F 01220      LD      C,DFH          ; MASK UNUSED CLOCK BITS
7E97 C0557F 01230      CALL    FILLER          ; GET HOURS LOW VALUE
7E9A 363A 01240      LD      [HL],3AH      ; PUT A COLON IN TIME$
7E9C 23 01250      INC      HL          ; BUMP THE STRING ALONG
7E9D 0602 01260      LD      B,2          ; NUMBER MINUTE/SEC L00PS
7E9F C0557F 01270      MINSEC CALL    FILLER          ; GET, CONVERT, SAVE VALUE
7EA2 C0557F 01280      CALL    FILLER          ; GET, CONVERT, SAVE VALUE
7EA5 363A 01290      LD      [HL],3AH      ; VALUE FOR A COLON
7EA7 23 01300      INC      HL          ; BUMP TIME BUFFER BY ONE
7EA8 10F5 01310      DJNZ    MINSEC      ; GO BACK FOR MIN/SEC
7EAA 28 01320      DEC      HL          ; BACK UP TO LAST COLON
7EAB 362D 01330      LD      [HL],20H      ; CHANGE TO STRING END
01340 ;
01350 ; #####
01360 ; HOURS, MINUTES, SECONDS ARE DONE ... NOW FIGURE AM/PM
01370 ; #####
01380 ;

7EAD 23 01390      INC      HL          ; BUMP TIME BUFFER BY ONE
7EAE F1 01400      POP      AF          ; GET BACK HOUR HI VALUE
7EAF C857 01410      BIT      2,A          ; CHECK AM/PM INDICATOR
7EB1 2804 01420      JR      Z,MORNNG      ; MORNING IF BIT 2 = 0
7EB3 365D 01430      LD      [HL],5DH      ; PUT LETTER "P" IN PLACE
7EB5 1802 01440      JR      NEXT          ; JUMP PAST LETTER A
7EB7 3641 01450      MORNNG LD      [HL],41H      ; PUT LETTER "A" IN PLACE
7EB9 23 01460      NEXT   INC      HL          ; BUMP TIME BUFFER BY ONE
7EBA 364D 01470      LD      [HL],40H      ; PUT LETTER "M" IN PLACE
7EBC C3842B 01480      JP      28B4H          ; BACK TO BASIC ACTIVITY
01490 ;
01500 ; #####
01510 ; THIS IS THE BEGINNING OF THE "CMD" PATCH TO SET TIME
01520 ; CHECK FOR TIME SETTING PARAMETERS AND SYNTAX
01530 ; #####
01540 ;

7EBF 7E 01550      START2 LD      A,[HL]      ; CHAR AT LINE POINTER
7EC0 FE22 01560      CP      22H          ; IS IT A QUOTE MARK?
7EC2 C29719 01570      JP      NZ,1997H      ; ?SN ERRDR IF NO QUOTE

```

## Bank Selecting Machine Language in ROM

**Warning:** Before you begin construction of anything in this section, read the rest of the chapter! You might want to construct the complete ROM/RAM bank select system.

Seriously, one of the most exciting aspects of the TRS-80 is its blank area in the memory map. This has been partially used in this chapter to install a real time clock. It can also be used to select machine language programs or data burned into ROMs. At the time of this book's publication, the cost of a 2K erasable, programmable, read only memory (EPROM) is less than \$8. A year earlier, when these memories were \$27 or more, this project would not have been practical. Now it is.

In quantities of 100, these EPROMs are less than \$5, which means, by using this project, direct access to over 200K of memory is possible for less than \$500. But one or two of such memories are just as valuable.

But first, what is an EPROM? How is it used? An EPROM, as its name suggests, is a memory which the user can program and reprogram as necessary. It is programmed with an EPROM programmer, and erased with ultraviolet light. It maintains its contents with the power off, just like the Level II ROMs themselves. It is used in a way even simpler than the way the TRS-80's RAM memory is used, and in this is found its great advantage: it needs but power, address and data lines to make its data available to the CPU.

By using a decoded output port, one of a bank of these memories may be selected for use. Here is an example; I might want to load a special machine language monitor program. I know that program is located in ROM #26 in my ROM bank. I can command something like -

```

OUT 31,26 <ENTER>
SYSTEM <ENTER>
/12288 <ENTER>

```

- and my program will be loaded and active. Only the time for three entries has been spent; not even the time of the disk access. And, beyond that, no RAM memory need be used!

A complete circuit for a ROM select bank is presented in Figure 8-14. Each of the lines marked 'to ROM' will select one of 256 possible ROMs!

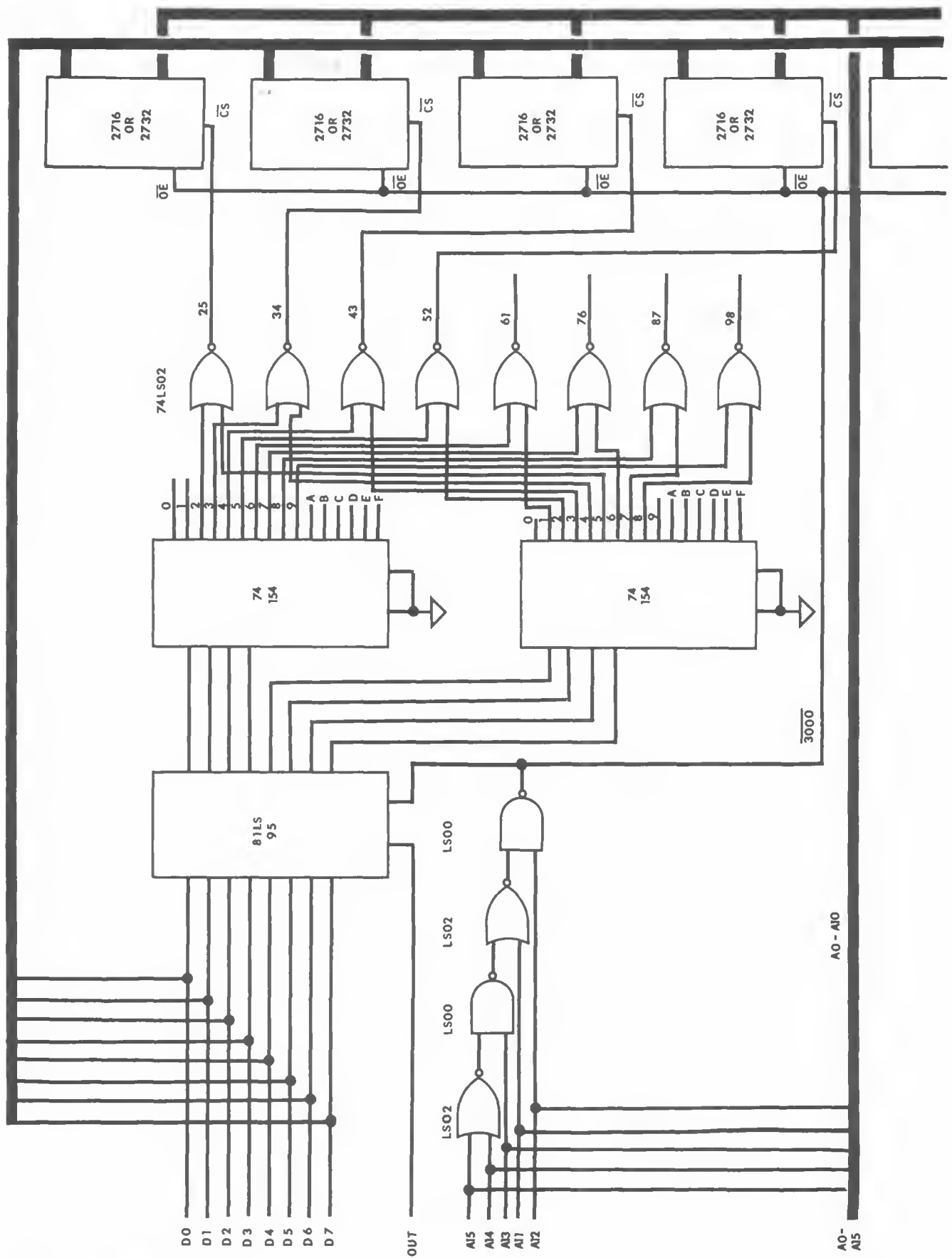


Figure 8-14. Bank-selected ROMs.

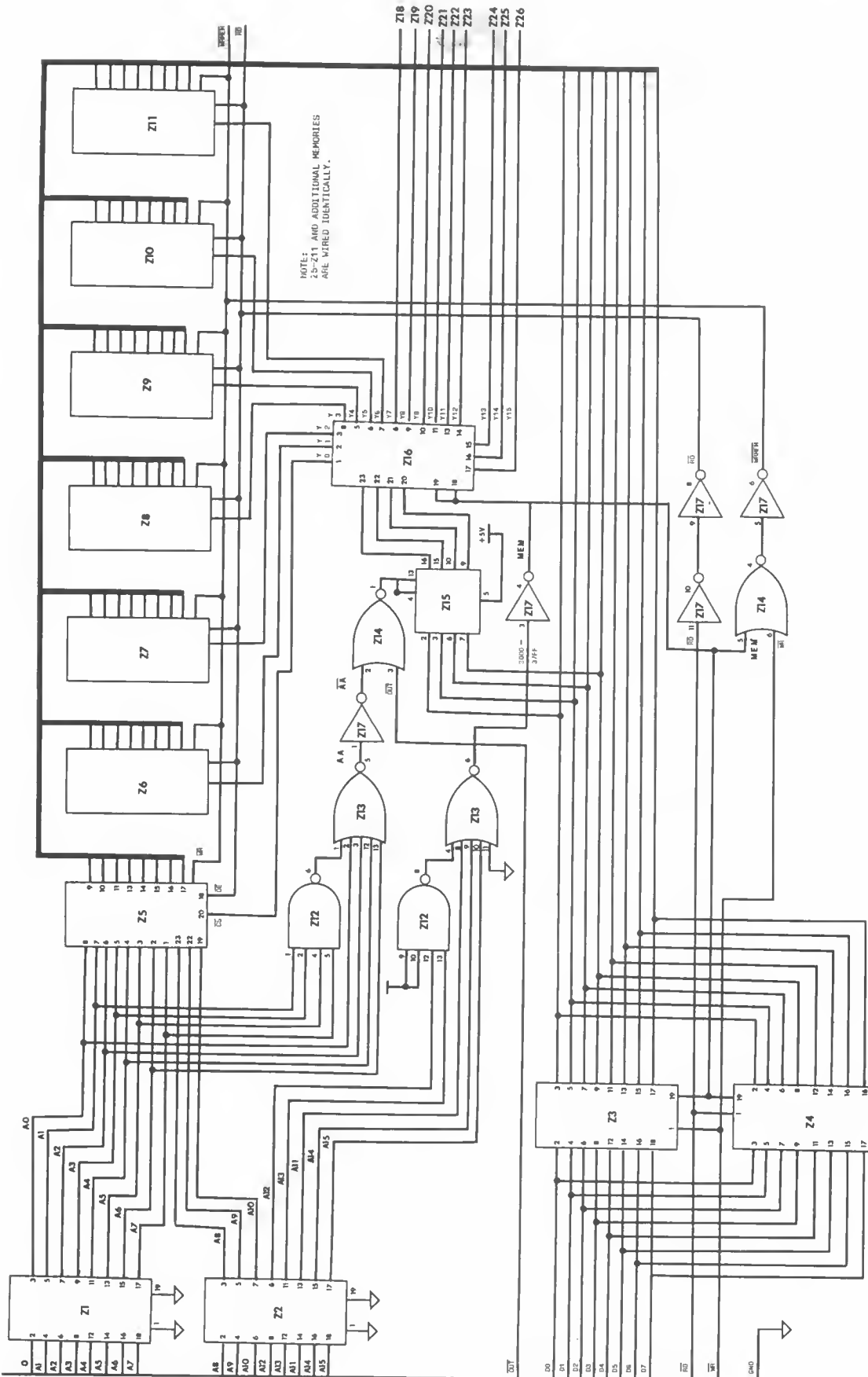


Figure 8-15. Bank-selected ROM/RAM.

## Bank Selecting ROMs

```

7EC5 23      01580      INC      HL      ; BUMP LINE PTR. BY ONE
7EC6 E5      01590      PUSH     HL      ; SAVE THE LINE POINTER
7EC7 11757F  01600      LD        DE, TABLE ; GET TABLE OF DAY NAMES
7ECA 0E00    01610      LD        C, 0      ; THIS WILL BE COUNTER
7ECC 0603    01620      OYLOOP   LD        B, 3      ; NUMBER OF CHARS IN DAY
7ECE E1      01630      POP       HL      ; GET LINE POINTER BACK
7ECF E5      01640      PUSH     HL      ; SAVE AGAIN FOR LOOP USE
7E00 1A      01650      FINDIT   LD        A, [0E]    ; GET 1ST CHAR OF STRING
7E01 A7      01660      AND       A      ; EASY WAY TO SET A FLAG
7E02 2808    01670      JR        Z, ERRORR1 ; VALUE = 0 ... ?SN ERROR
7E04 8E      01680      CP        (HL)    ; CHECK IT AGAINST TABLE
7E05 280A    01690      JR        Z, GOTONE   ; GET READY FOR NEXT CHAR
7E07 13      01700      LOOP4    INC       DE      ; RUN PAST VALUES FOR DAY
7E08 10F0    01710      DJNZ     LD, DP4   ; 8Y RUNNING B TO ZERO
7E0A 0C      01720      INC       C      ; NEXT DAY - BUMP COUNTER
7E08 18EF    01730      JR        OYLOOP   ; BACK TO NEXT DAY LOOP
7E00 E1      01740      ERRORR1 POP      HL      ; CLEAR STACK OF HL REG.
7E0E C39718  01750      ERRORR2 JP       1997H ; GO TO ?SN ERROR MESSAGE
7E11 23      01760      GOTONE   INC       HL      ; GET NEXT CHAR FROM LINE
7E12 13      01770      INC       DE      ; BUMP TABLE VALUE ALONG
7EE3 10E8    01780      DJNZ     FINDIT   ; KEEP GOING TILL DONE
7E1790 ;
01800 ; #####
01810 ; NUMERICAL VALUE FOR DAY IS IN C - PUT IT IN MSM5832
01820 ; #####
01830 ;
7EE5 F1      01840      POP       AF      ; CLEAR STACK OF HL VALUE
7EE6 F0360380 01850      LD        (IY+3), B0H ; SET UP B255 TO WRITE
7EEA C06A7F  01860      CALL     DELAY      ; THAT #&1% SLOW MSM5832
7EE0 F0360250 01870      LD        (IY+2), 50H ; CLOCK CHIP WRITE VALUE
7EF1 C06A7F  01880      CALL     DELAY      ; HOW SLOW IS IT?
7EF4 78      01880      LD        A, C      ; GET DAY OF WEEK VALUE
7EF5 F0360106 01900      LD        (IY+1), 6   ; READY TO WRITE DAY
7EF9 C06A7F  01910      CALL     DELAY      ; WAIT (YAWN) TO WRITE
7EFC F07700  01920      LD        (IY+0), A   ; WRITE DAY TO CLOCK
01930 ;
01940 ; #####
01950 ; DAY IS WRITTEN - FIND MONTH, DAY, YEAR AND WRITE THEM
01960 ; #####
01970 ;
7EFF 1808    01980      LD        0, 11      ; VALUE FOR MONTH + 1
7F01 C0377F  01990      CALL     TIMSET      ; WRITE MONTH TO CLOCK
7F04 C0377F  02000      CALL     TIMSET      ; WRITE DAY TO CLOCK
7F07 1600    02010      LD        0, 13      ; VALUE FOR YEAR +1
7F08 C0377F  02020      CALL     TIMSET      ; WRITE YEAR TO CLOCK
7F0C 1605    02030      LD        0, 5       ; SET TO HOURS HIGH VALUE
7F0E C01A7F  02040      CALL     AMORPM      ; WRITE HOURS TO CLOCK
7F11 C0377F  02050      CALL     TIMSET      ; WRITE MINUTES TO CLOCK
7F14 0804    02060      LD        8, 4       ; NUMBER OF CHARS LEFT
7F16 23      02070      SNEAK   INC       HL      ; BUMP LINE POINTER
7F17 10F0    02080      DJNZ     SNEAK      ; LOOP PAST "PM" & QUOTES
7F19 C9      02080      RET              ; BACK TO BASIC PROGRAM
02100 ;
02110 ; #####
02120 ; CHECK FOR AM OR PM INDICATION AND WRITE THAT VALUE
02130 ; #####
02140 ;
7F1A 23      02150      AMORPM   INC       HL      ; BUMP LINE TO NEXT CHAR.
7F18 E5      02160      PUSH     HL      ; SAVE CURRENT LINE PTR.
7F1C 05      02170      PUSH     DE      ; SAVE OTHER VALUES IN DE
7F1D 110600  02180      LD        DE, 6      ; HOW MANY SPACES TO MOVE
7F20 19      02190      AOO      HL, 0E      ; FIND AM OR PM IN LINE
7F21 7E      02200      LD        A, (HL)    ; GET CHARACTER FROM LINE
7F22 FE41    02210      CP        41H      ; SET FLAG IF CHAR. = "A"
7F24 3E04    02220      LD        A, 4       ; GET PM INDICATOR READY
7F26 20D1    02230      JR        NZ, EVENNG ; ZERO FLAG NOT SET IF PM
7F28 AF      02240      XOR       A         ; CLEAR PM INDICATOR
7F29 01      02250      EVENNG   POP      DE      ; RESTORE VALUES TO DE
7F2A E1      02260      POP      ML      ; GET ORIGINAL LINE PTR.
7F2B 0602    02270      LD        B, 2       ; SET UP 8 AS TIMSET LOOP
7F2D 4F      02280      LD        C, A       ; SAVE AM/PM INDICATOR
7F2E 7E      02290      LD        A, (HL)    ; GET VALUE FROM LINE
7F2F 0630    02300      SUB      30H      ; STRIP ASCII MASK
7F31 38AA    02310      JR        C, ERRORR1 ; ERROR IF LESS THAN 0
7F33 81      02320      AOO      A, C       ; ADD AM/PM BIT TO VALUE
7F34 C3417F  02330      JP        MIDDLE    ; SUBROUTINE FINISHES JOB
02340 ;
02350 ; #####
02360 ; TIME SETTING SUBROUTINE CHECKS LINE FOR SYNTAX
02370 ; #####
02380 ;
7F37 1E30    02390      TIMSET   LD        E, 30H ; CONVERTS ASCII TO HEX
7F39 0602    02400      LD        B, 2       ; LOOP TWICE FOR 2 DIGITS
7F3B 16      02410      ZLOOP    DEC       D      ; BUMP CLOCK ADDRESS PORT
7F3C 23      02420      INC       HL      ; GET NEXT CHAR FROM LINE
7F3D 7E      02430      LD        A, (ML)    ; MOVE IT TO ACC. TO TEST
7F3E 83      02440      SUB      E         ; STRIP OFF ASCII VALUE
7F3F 3B9C    02450      JR        C, ERRORR1 ; ERROR IF LESS THAN 0
7F41 FE0A    02460      MIDDLE   CP        DAM      ; CHECK IF GREATER THAN 8
7F43 308B    02470      JR        NC, ERRORR1 ; ERROR IF GREATER THAN 9
7F45 F07201  02480      LD        (IY+1), 0   ; OPEN PORT TO CLOCK
7F48 C08A7F  02480      CALL     DELAY      ; THE USUAL CMOS WAIT
7F4B F07700  02500      LD        (IY+0), A   ; WRITE VALUE TO CLOCK
7F4E C08A7F  02510      CALL     DELAY      ; WAIT! WAIT! WAIT!!!!
7F51 10EB    02520      DJNZ     ZLOOP      ; DO IT FOR 2 DIGITS
7F53 23      02530      INC       HL      ; BUMP PAST / : OR SPACE
7F54 C8      02540      RET              ; BACK TO MAIN PROGRAM
02550 ;
02560 ; #####
02570 ; GET VALUE, CONVERT TO ASCII, AND SAVE IN TIMES BUFFER
02580 ; #####
02590 ;

```

Actually, the bank-select ROM idea is only one part of a larger possibility — blocks of ROM and/or RAM placed interchangeably. Ideally the additional RAM would be available in higher memory, but the TRS configuration is so locked into its memory map that expansion in high memory would require major reconfiguring of the memory and refresh circuits. With that in mind, then, all this expansion will be dedicated to the free area located at 3000 to 37E0.

This memory addition is an application of a 'Read-Only-RAM' concept. Simply stated, the memory write (WR) line to read/write memory is disabled by the user in order to outline an area of protected, but not permanently programmed, memory. Machine language programs under development can be emulated with this system. Crucial software can be embedded in crash-proof RAM, and the occasional nuisance of a program gone wild will not affect data in this area.

The inclusion of bank selection in this area permits the use of interchangeable, and constantly on-line, blocks of ROM, RAM, and protected RAM. In Figures 8-16 and 8-17, the ROMs used, as before, are 2716 EPROMs, and the RAMs are of the static variety. Two designs of the circuit are shown: one uses 2114 static RAMs, which are 1K by 4 bits wide; the other uses 4118 static RAMs, each a full 1K by 8 bits wide. The former memories are considerably less expensive, but the latter are easier to wire in a wire-crazy bank select scheme like this one.

As before, output port 31 has been chosen for the memory, and each 2K block is selected by the value output through port 31. Furthermore, RAM data in the unselected blocks remains intact, ready to use whenever a different value is output through port 31.

The bank-selected ROM may be used for the most part as any other memory, with one very interesting exception: routines in *different* ROMs may not call or jump to each others' routines in the normal way. Instead, two special routines must be used.

The jump routine is the simplest: the ROM's position in the bank must be identified, the jump prepared and the OUT statement commanded. The easiest way to do this is shown in Listing 8-7; the AF and HL registers are saved on the stack, then AF is loaded with the location of the routine (which it must obtain from a table of some sort identical in each ROM). Following that, the HL register pair is loaded with the jump address, and the OUT (1F), A command is then

```

7F55 15      02600 FILLER DEC 0 ; BUMP CLOCK PORT ADDRESS
7F56 F07201 02610 LO [IY+1],0 ; POINT TO VALUE WANTED
7F58 C06A7F 02620 CALL DELAY ; THAT OL' SLOW MSM5832
7F5C F07E00 02630 LO A,[IY+0] ; GET DUMMY VALUE INTO A
7F5F C06A7F 02640 CALL DELAY ; WAIT AGAIN! SLOW CHIP
7FB2 F07E00 02650 LO A,[IY+0] ; NOW GET VALID VALUE
7FB5 A1      02660 AND C ; MASK UNUSED BITS
7FB8 83      02670 AOR A,E ; MAKE IT AN ASCII VALUE
7FB7 77      02680 LO [HL],A ; PUT VALUE INTO BUFFER
7FB8 23      02680 INC HL ; NEXT BUFFER POSITION
7FB8 C8      02700 RET ; BACK TO MAIN PROGRAM

02710 ;
02720 ; #####
02730 ; THIS IS A SETUP WHICH CALLS A DELAY SUBROUTINE IN ROM
02740 ; #####
02750 ;
7F6A C5      02760 DELAY PUSH BC ; SAVE BC REGISTER PAIR
7F88 F5      02770 PUSH AF ; SAVE AF REGISTER PAIR
7F6C 010100 02780 LO BC,1 ; DELAY FOR MSM5832 CHIP
7FBF C06000 02790 CALL 0060H ; HERE IS ROUTINE IN ROM
7F72 F1      02800 POP AF ; GET AF REGISTERS BACK
7F73 C1      02810 POP BC ; GET BC REGISTERS BACK
7F74 C8      02820 RET ; BACK TO MAIN PROGRAM

02830 ;
02840 ; #####
02850 ; THIS IS THE LOOKUP TABLE OF DAYS OF THE WEEK
02860 ; #####
02870 ;
7F75 40      02880 TABLE DEFB 'MON'
02890 ;
7F78 54      02900 DEFB 'TUE'
02910 ;
7F78 57      02920 DEFB 'WED'
02930 ;
7F7E 54      02940 DEFB 'THU'
02950 ;
7FB1 46      02960 DEFB 'FRI'
02970 ;
7FB4 53      02980 DEFB 'SAT'
02990 ;
7FB7 53      03000 DEFB 'SUN'
03010 ;
7F8A 00      03020 DEFB 0
03030 ;
03040 ; #####
7E00 03050 END ENTRY
00000 TOTAL ERRORS

```

executed, which switches ROMs. The proper jump address is still in HL, so a simple JP (HL) effects the jump.

```

; ROUTINE BEING LEFT MUST PROVIDE THIS
; INFORMATION TO AF AND HL REGISTERS
..ZZZZ LO (ZZZZ),A ; SAVE AF VALUES
..YYYY LO (YYYY),HL ; SAVE HL VALUES
3ENN LO A,NN ; NEW ROM BANK NUMBER
21NNNN LO HL,NNNN ; JUMP ADDRESS
C3WWW JP XFER1 ; MAKE ROM TRANSFER
; ALL ROMS CONTAIN THE FOLLOWING IDENTICAL
; BYTES AT THE SAME ADDRESSES IN ROM
031F XFER1 OUT (1F),A ; SWITCHES ROMS
E9 JP [HL] ; JUMPS TO ROUTINE
..ZZZZ LO (ZZZZ),A ; NOT USED IN JUMP
..XXXX LO A,(XXXX) ; NOT USED IN JUMP
031F OUT (1F),A ; NOT USED IN JUMP
C9 RET ; NOT USED IN JUMP
; ALL ROUTINES BEING ENTERED MUST PROVIDE
; THE FOLLOWING RESTORATION CODING
..ZZZZ LO HL,(ZZZZ) ; RESTORE HL VALUES
..YYYY LO A,(YYYY) ; RESTORE AF VALUES

```

Listing 8-6. Accessing multiple ROMs (jumps).

Calling a subroutine in another ROM is more complicated, but still can be done. Listing 8-(?) shows how it might be achieved.

```

; ROUTINE BEING LEFT MUST PROVIDE THIS
; INFORMATION IN AF AND HL REGISTERS
..ZZZZ LO (ZZZZ),A ; SAVE AF DATA
..YYYY LO (YYYY),HL ; SAVE HL DATA
DB1F IN A,(1F) ; GET ROM BANK NUMBER
..XXXX LO (XXXX),A ; SAVE ROM BANK NO.
21NNNN LO HL,NNNN ; GET CALL ADDRESS
C0WWW CALL XFER2 ; MAKE ROM TRANSFER
; THE FOLLOWING CALLING ROUTINE MUST BE
; PLACED IN ALL ROMS AT IDENTICAL LOCATION
031F XFER2 OUT (1F),A ; SWITCH ROMS
E9 JP [HL] ; ENTER ROUTINE
..ZZZZ BACK LO (ZZZZ),A ; SAVE AF DATA
..XXXX LO A,(XXXX) ; GET OLD ROM NO.
031F OUT (1F),A ; TRANSFER BACK
C9 RET ; BACK TO CALLER
; THE CALLED ROUTINE MUST MAKE THE
; FOLLOWING IDENTIFICATIONS
..YYYY LO HL,(YYYY) ; GET SAVED DATA
..ZZZZ LO A,(ZZZZ) ; GET SAVED DATA
; EXECUTE SUBROUTINE FOUND HERE
; PERFORM FOLLOWING STEPS WHEN THE
; SUBROUTINE IS COMPLETE AND MUST RETURN
C3 VVVV JP BACK ; GO TO XFER ROUTINE

```

Listing 8-7. Accessing multiple ROMs (calls).

This bank-select method is only one of many options which may be selected to move from one ROM to another; as you can see, three bytes of RAM (marked ZZZZ, YYYY and WWWW in the listings) are needed to store information between transfers. Although this process may initially seem unwieldy, a ROM-resident operating system in each one, together with cautious programming, will provide a remarkably transparent system expansion.

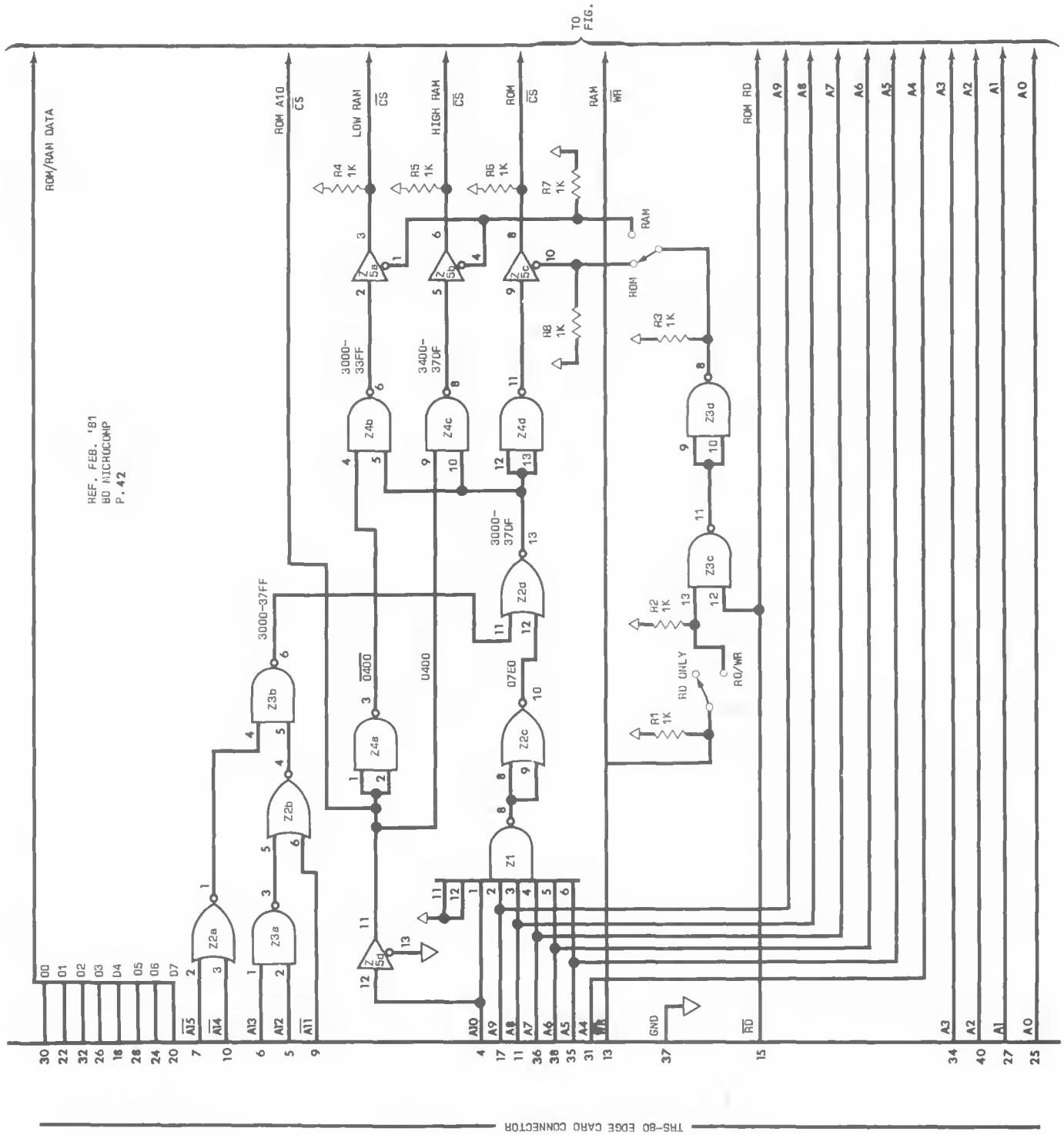


Figure 8-16. Bank-selected 4118 ROM/RAM.

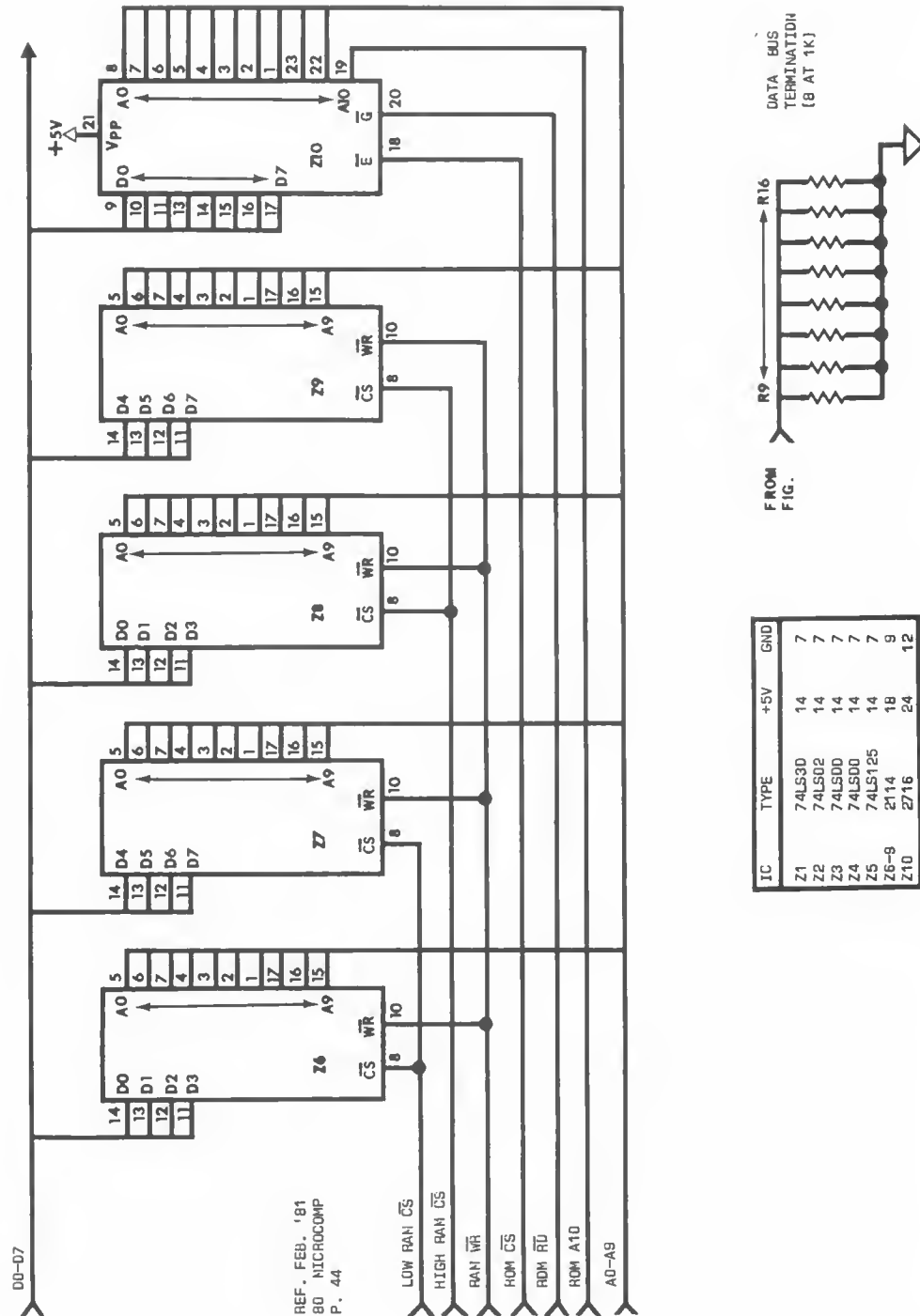
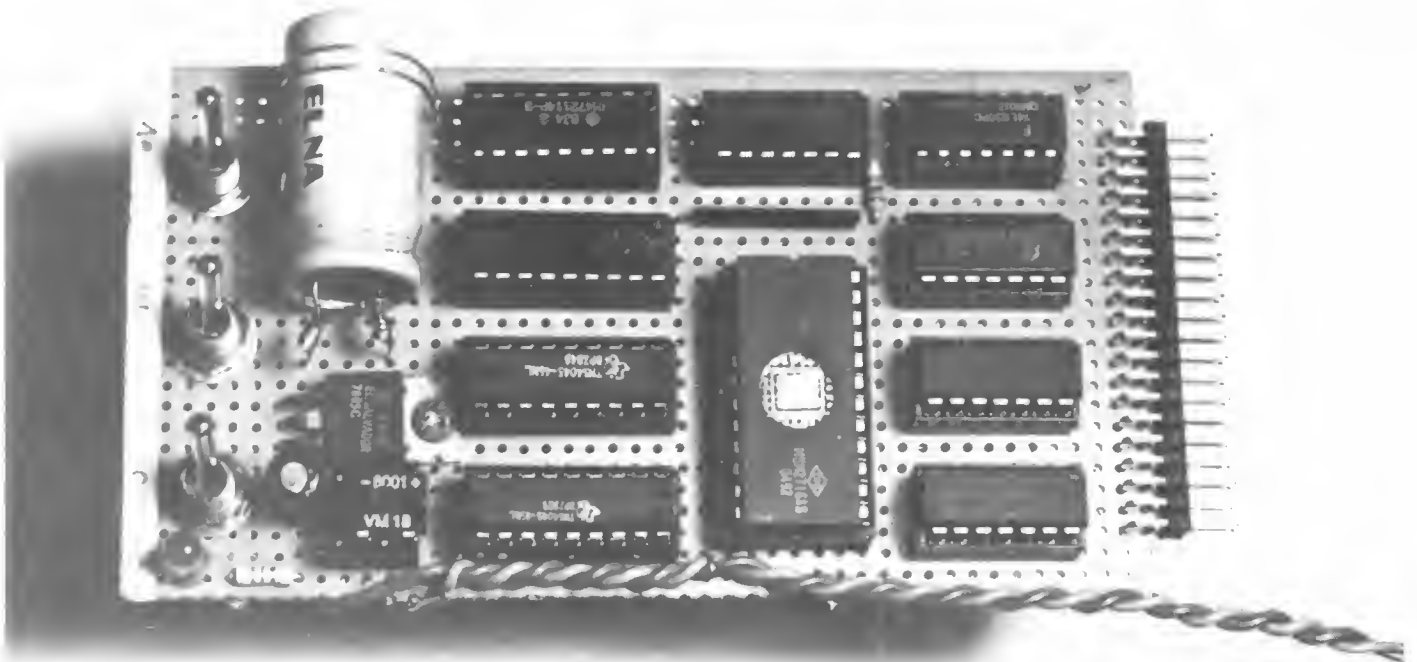
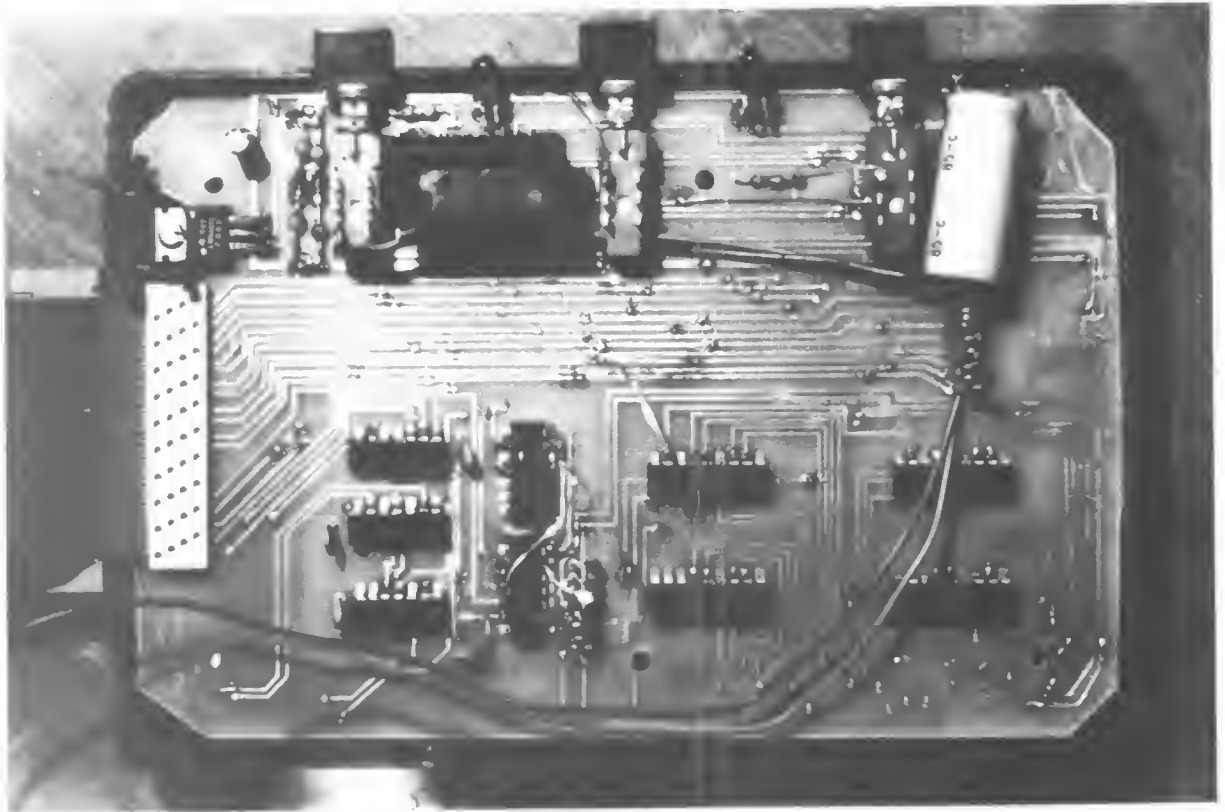


Figure 8-17. ROM/RAM addition without bank-select.



*Completed ROM/RAM addition with power supply on board.*

```

00100 ; MEMORY SIDECAR TEST TAPE (C) 1981 D. B. KITSZ
00110 ;
3C00          00120      ORG      3C00H
3C00 2A       00130      DEFW    '** LOADING 2K RAM TEST TAPE '
3C1C 2D       00140      DEFW    '— D. B. KITSZ, SEPTEMBER 1981 **'
3000          00150      DRG      3000H
3000 2A       00160 M1      DEFW    '*****'
3020 2A       00170      DEFW    '*****'
3040 2D       00180 M2      DEFW    '----- TESTING "READ-ONLY" '
3060 2D       00190      DEFW    '—RAM" MEMORY AREA -----'
3080 54       00200 M3      DEFW    'THIS SCREEN HAS BEEN LOADED OIRE'
30A0 43       00210      DEFW    'CTLY FROM THE MEMORY SIDECAR. IF'
30C0 41       00220 M4      DEFW    'ALL CHARACTERS ARE CDRRECT IN TH'
30E0 49       00230      DEFW    'IS MESSAGE, AND IT HAS LOADED ON'
3100 54       00240 M5      DEFW    'THE SCREEN NORMALLY, THEN THIS F'
3120 49       00250      DEFW    'IRST TEST OF YOUR MEMORY SIDECAR'
3140 49       00260 M6      DEFW    'IS COMPLETED. BELOW IS SHOWN A '
3160 43       00270      DEFW    'COMPLETE CHARACTER SET AVAILABLE'
3180 54       00280 M7      DEFW    'TO YOUR TRS-80: '
31A0 2D       00290      DEFW    '
31C0 0001     00300 M8      DEFW    010DH
31C2 0203     00310      DEFW    0302H
31C4 0405     00320      DEFW    0504H
31C6 0607     00330      DEFW    0706H
31C8 0809     00340      DEFW    0908H
31CA 0A0B     00350      DEFW    0B0AH
31CC 0C0D     00360      DEFW    0D0CH
31CE 0E0F     00370      DEFW    0F0EH
3100 1011     00380      DEFW    1110H
3102 1213     00390      DEFW    1312H
3104 1415     00400      DEFW    1514H
3106 1617     00410      DEFW    1716H
3108 1819     00420      DEFW    1918H
310A 1A1B     00430      DEFW    1B1AH
310C 1C1D     00440      DEFW    1D1CH
310E 1E1F     00450      DEFW    1F1EH
3110 2021     00460      DEFW    2120H
3112 2223     00470      DEFW    2322H
3114 2425     00480      DEFW    2524H
3116 2627     00490      DEFW    2726H
3118 2829     00500      DEFW    2928H
311A 2A2B     00510      DEFW    2B2AH
311C 2C2D     00520      DEFW    2D2CH
311E 2E2F     00530      DEFW    2F2EH
3120 3031     00540      DEFW    3130H
3122 3233     00550      DEFW    3332H
3124 3435     00560      DEFW    3534H
3126 3637     00570      DEFW    3736H
3128 3839     00580      DEFW    3938H
312A 3A3B     00590      DEFW    3B3AH
312C 3C3D     00600      DEFW    3D3CH
312E 3E3F     00610      DEFW    3F3EH
3200 4041     00620 M9      DEFW    4140H
3202 4243     00630      DEFW    4342H
3204 4445     00640      DEFW    4544H
3206 4647     00650      DEFW    4746H
3208 4849     00660      DEFW    4948H
320A 4A4B     00670      DEFW    4B4AH
320C 4C4D     00680      DEFW    4D4CH
320E 4E4F     00690      DEFW    4F4EH
3210 5051     00700      DEFW    5150H
3212 5253     00710      DEFW    5352H
3214 5455     00720      DEFW    5554H
3216 5657     00730      DEFW    5756H
3218 5859     00740      DEFW    5958H
321A 5A5B     00750      DEFW    5B5AH
321C 5C5D     00760      DEFW    5D5CH
321E 5E5F     00770      DEFW    5F5EH
3220 6061     00780      DEFW    6160H
3222 6263     00790      DEFW    6362H
3224 6465     00800      DEFW    6564H
3226 6667     00810      DEFW    6766H
3228 6869     00820      DEFW    6968H
322A 6A6B     00830      DEFW    6B6AH
322C 6C6D     00840      DEFW    6D6CH
322E 6E6F     00850      DEFW    6F6EH
3230 7071     00860      DEFW    7170H
3232 7273     00870      DEFW    7372H
3234 7475     00880      DEFW    7574H
3236 7677     00890      DEFW    7776H
3238 7879     00900      DEFW    7978H
323A 7A7B     00910      DEFW    7B7AH
323C 7C7D     00920      DEFW    7D7CH
323E 7E7F     00930      DEFW    7F7EH
3240 8081     00940 M10     DEFW    8180H
3242 8283     00950      DEFW    8382H
3244 8485     00960      DEFW    8584H
3246 8687     00970      DEFW    8786H
3248 8889     00980      DEFW    8988H
324A 8A8B     00990      DEFW    8B8AH
324C 8C8D     01000      DEFW    8D8CH
324E 8E8F     01010      DEFW    8F8EH

```

Remember also that the ROMs do not need to call each other directly. Instead, you may want to establish a lookup and transfer table in RAM, that acts like this :

1. User is in BASIC.
2. OUT 31,0 is entered to select the master ROM.
3. SYSTEM is entered, followed by /12288.
4. The master ROM (ROM #0) may be programmed to reset MEMORY SIZE and relocate a bank of patch points into RAM.
5. The master ROM, having completed its work the way Level II does at power up, returns to a READY in BASIC.
6. The routines are now all ready to use.

In summary, the ROM, RAM and bank-select systems can extend the horizons of your TRS-80 in unique ways. A project for people afflicted with cerebral palsy is developing a system whereby patients, doctors, or nurses need not be concerned if the TRS-80 crashes at some point, or the power is removed. Instead of loading tapes or fumbling with disks that may get damaged through handling or erratic power, these people need only use their bank-selected ROMs to reinstate and activate the machine language programs which drive specially made hardware. This hardware permits them to communicate with and use the computer with great ease.

Without the bank-select feature, however, the 8K control programs would have to be reloaded regularly, at a cost of time and patience, plus the risk of damage, loss or failure.

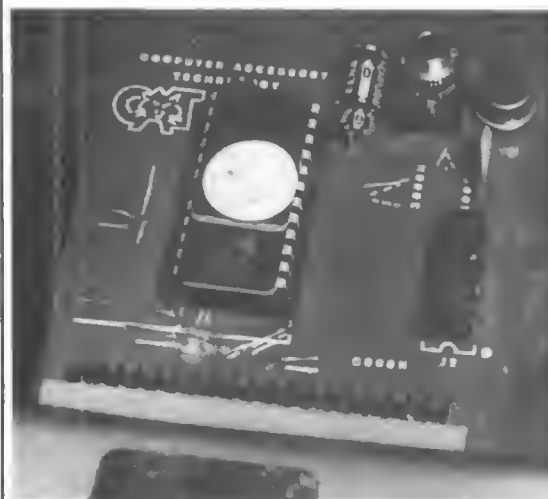
## Continued Listing

3250	90B1	D1020	DEFW	9190H
3262	8293	D1030	DEFW	93B2H
3264	84B5	D1040	DEFW	95B4H
3266	86B7	D1050	DEFW	97B6H
3268	88B9	D1060	DEFW	99B8H
326A	8AB8	D1070	DEFW	9BABH
326C	8CB0	D1080	DEFW	9D9CH
326E	8E8F	D1090	DEFW	9F8BH
326D	ADA1	D1100	DEFW	9A1ADH
3282	A2A3	D1110	DEFW	9A3A2H
3264	A4A5	D1120	DEFW	9A5A4H
3266	A6A7	D1130	DEFW	9A7ABH
3268	ABA8	D1140	DEFW	9ABABH
326A	AAAB	D1150	DEFW	9ABAAH
326C	ACAD	D1160	DEFW	9ADACH
326E	AEAF	D1170	DEFW	9AFABH
327D	8D81	D1180	DEFW	9B190H
3272	8293	D1190	DEFW	9B3B2H
3274	84B5	D1200	DEFW	9B594H
3276	86B7	D1210	DEFW	9B78BH
3278	88B8	D1220	DEFW	9B98BH
327A	8AB8	D1230	DEFW	9BB8AH
327C	8CB0	D1240	DEFW	9BD8CH
327E	8E8F	D1250	DEFW	9BF8BH
328D	C0C1	D1260	DEFW	9C1CDH
3282	C2C3	D1270	DEFW	9C3C2H
3284	C4C5	D1280	DEFW	9C5C4H
3286	C6C7	D1290	DEFW	9C7C6H
3288	C8C9	D1300	DEFW	9C9C8H
328A	CACB	D1310	DEFW	9CBCAH
328C	CEC0	D1320	DEFW	9CCECH
328E	CECF	D1330	DEFW	9CFCEH
3290	D0D1	D1340	DEFW	9D1DDH
32B2	D2D3	D1350	DEFW	9D3DDH
3284	D4D5	D1360	DEFW	9D5D4H
3286	D6D7	D1370	DEFW	9D7D6H
3288	D8D9	D1380	DEFW	9D9D8H
328A	DADE	D1390	DEFW	9DADAH
328C	DCDD	D1400	DEFW	9DDDCCH
328E	DEDF	D1410	DEFW	9DDFDBH
32AD	EDF1	D1420	DEFW	9E1EDH
32A2	E2E3	D1430	DEFW	9E3E2H
32A4	E4E5	D1440	DEFW	9E5E4H
32A6	E6E7	D1450	DEFW	9E7E6H
32A8	E8E9	D1460	DEFW	9E9E8H
32AA	EAB8	D1470	DEFW	9EBAEH
32AC	ECED	D1480	DEFW	9EDECH
32AE	EEEF	D1490	DEFW	9EFEBH
3280	FD1	D1500	DEFW	9F1FDH
3282	F2F3	D1510	DEFW	9F3F2H
3284	F4F5	D1520	DEFW	9F5F4H
3286	F6F7	D1530	DEFW	9F7F6H
3288	F8F9	D1640	DEFW	9F9F8H
328A	FAF8	D1650	DEFW	9FBFAH
328C	FCFD	D1660	DEFW	9FDFCH
328E	FEFF	D1670	DEFW	9FFFBH
32C0	20	D1580	DEFW	
32E0	20	D1590	DEFW	
3300	57	D1600	DEFW	
3320	43	D1610	DEFW	
3340	43	D1620	DEFW	
336D	48	D1630	DEFW	
3380	20	D1640	DEFW	
33A8	20	D1650	DEFW	
33CD	2A	D1660	DEFW	
33E0	2A	D1670	DEFW	
340D	2A	D1680	DEFW	
342D	2A	D1690	DEFW	
3440	20	D1700	DEFW	
346D	20	D1710	DEFW	
3471	54	D1720	DEFW	
3491	20	D1730	DEFW	
34B1	32	D1740	DEFW	
34D1	20	D1750	DEFW	
34F1	53	D1760	DEFW	
3511	4F	D1770	DEFW	
3531	47	D1780	DEFW	
3551	44	D1790	DEFW	
3571	20	D1800	DEFW	
3575	211640	D1810	ENTRY	LD
357B	36E3	D1820		LD
357A	23	D1830		INC
357B	36D3	D1840		LD
357D	210D30	D1850	BACKUP	LD
368D	110D3C	D1860		LD
3583	010D04	D1870		LD
3586	ED8D	D1880		LDIR
358B	3A403B	D1890	LOOP1	LD
358B	A7	D1900		AND
358C	28FA	D181D		JR
358E	CDA435	D182D		CALL
3591	2634	D183D		LD
35B3	163C	D184D		LD

## Romplus and Other ROM Extenders

There are simpler methods of adding ROM to the TRS-80, where some of the work has already been done for you. The *Micro 80 Computer Club of Ontario* (in care of Brian Harron, 67-3691 Albion Road, Ottawa, Ontario K1T 1P2) has produced their Romplus board, capable of handling two 2708 1K ROMs or one 2716 2K ROM. It fits inside the keyboard, and only requires soldering three integrated circuits in place on the board, plus sockets for the ROMs.

*Computer Accessory Technology* (1307 Bagley Drive, Kokomo, Indiana 46901) has also developed a small board which contains a single 2716 EPROM and an address decoder. It comes caseless, but has a power supply and plugs directly into the TRS-80 edge-card connector. A set of programmed ROMs is available which include utilities of different kinds.



*Personal Micro Computers* (475 Ellis Street, Mountain View, California 94043) makes the REX-80 ROM extender, which is similar to the C.A.T. board, but comes with case and power supply, plus edge connector and cable. They also provide programmed ROMs. About \$50 for the device, \$25 for the ROMs.

Finally, *The Peripheral People* (P.O. Box 524, Mercer Island, Washington 98040), offers the Memory Sidecar ROM/RAM addition, which I designed, and is identical to the ROM/RAM addition presented in this book, without any bank-select features. The complete unit is \$149, and a blank board is \$25.

## Continued Listing

```

3595 0604 01950 LD 8,4H
3597 E080 01960 LDIR
3599 3A403B 01970 LDDP2 LD A,(3B40H)
359C A7 01980 AND A
3580 28FA 01990 JR Z,LDDP2
359F CDA435 02000 CALL DELAY
35A2 1809 02010 JR BACKUP
35A4 010040 02020 DELAY LD BC,4000H
35A7 0B 02030 LDDP3 DEC BC
35A8 78 02040 LD A,B
35A8 81 02050 OR C
35AA 20FB 02060 JR NZ,LDDP2
35AC C9 02070 RET
35AD 20 02080 DEFM ' '
35B1 20 02090 M8A DEFM '-----'
3501 20 02100 DEFM '-----'
35F1 20 02110 M9A DEFM ' - WHICH IS THE ACTUAL BLOCK'
3611 20 02120 DEFM ' OF MACHINE CODE THAT IS RUNNING'
3631 54 02130 M1DA DEFM 'THIS TEST PROGRAM. TO EXIT THIS'
3651 20 02140 DEFM ' ROUTINE, YOU MUST PRESS RESET.'
3871 54 02150 M11A DEFM 'TO REPEAT THE TEST SEQUENCE, PRE'
3691 53 02160 DEFM 'SS THE <ENTER> KEY.'
3681 20 02170 M12A DEFM ' '
3601 20 02180 DEFM ' '
36F1 54 02190 M13A DEFM 'THE UNUSUAL LINES SEEN HERE ARE '
3711 41 02200 DEFM 'A REPRESENTATION OF MEMORY SPACE'
3731 55 02210 M14A DEFM 'USED BY THE TRS-80 TO CONTROL TH'
3751 45 02220 DEFM 'E DISK DRIVES AND CASSETTE PORT:'
3771 20 02230 DEFM ' '
3791 20 02240 DEFM ' '
37B1 20 02250 DEFM ' '
3C40 02260 ORG 3C40H
3C40 2A 02270 DEFM '** LOADING 2K RAM TEST TAPE COM'
3C5F 50 02280 DEFM 'PLETE. CONTROL TAKEN BY TEST **'
4016 02290 ORG 4016H
4016 7535 02300 DEFW ENTRY
3575 02310 END ENTRY
00000 TOTAL ERRORS
29944 TEXT AREA BYTES LEFT

```

```

BACKUP 3570 01850 02010
DELAY 35A4 02020 01920 02000
ENTRY 3575 01810 02300 02310
LDDP1 3588 01890 01810
LDDP2 3599 01970 01990
LDDP3 35A7 02030 02060
M1 3000 00160
M10 3240 00940
M10A 3631 02130
M11 3280 01260
M11A 3671 02150
M12 32C0 01580
M12A 3681 02170
M13 3300 01600
M13A 36F1 02190
M14 3340 01620
M14A 3731 02210
M15 3380 01640
M16 33C0 01660
M1A 3400 01680
M2 3040 00180
M2B 3440 01700
M3 3080 00200
M3A 3471 01720
M4 30C0 00220
M4A 3481 01740
M5 3100 00240
M5A 34F1 01760
M6 3140 00260
M6A 3531 01780
M7 3180 00280
M7A 3571 01800
M8 31C0 00300
M8A 35B1 02090
M9 3200 00620
M9A 35F1 02110

```

## A Front Panel Monitor

The first personal computers were a hobbyist's dream and a user's nightmare. These large boxes of electronic boards were programmed by hand, one byte at a time. The operation of the processor was stopped, an address was selected, and a byte programmed — all by using nearly 30 switches.

The TRS-80 is a far cry in size, speed, power, and convenience from these early machines (but don't forget that 'early' means 1974!). Yet there was an advantage in these early machines that the TRS-80 and its kin don't have: the front panel display. The front panel not only contained the multiple switches, but also a bank of LEDs so the user could view the contents of memory, registers, etc. (see Photo 8-2. ).

The front panel is not an entirely obsolete concept, and can be remarkably valuable when your machine language programs sprint for the exit when you're not looking. Since the front panel visually monitors addresses and data, it provides somewhat of a window opening on the computer's activities.

The front panel can tell you if the machine is caught up in a deadly tight loop, if it is still processing (have you ever waited while the computer sorted string data?), or if it is operating in the area you expect it to. You can follow peripheral accesses like printers and disk, sound output routines and special devices you may have created.

Figure 8-18 presents the circuit for the micro front panel. It is no more than a group of latches which are activated by certain conditions — you may select any combination of input, output, read or write signals to trigger the LED displays. 16 LEDs monitor the address, and eight monitor the data.

The latches are triggered on an upswing of any signal line that is switched into the select gate; the gate may be switched off entirely, leaving the last latched address displayed on the LEDs.

Ideally, the front panel can be created from subminiature, 'grain of wheat' LEDs, and mounted directly in the TRS-80 case and soldered in place. Alternatively, it may be connected when needed by a standard 40 conductor cable. Method of construction is not critical, and can use soldering and wire wrap.

## A Front Panel Monitor

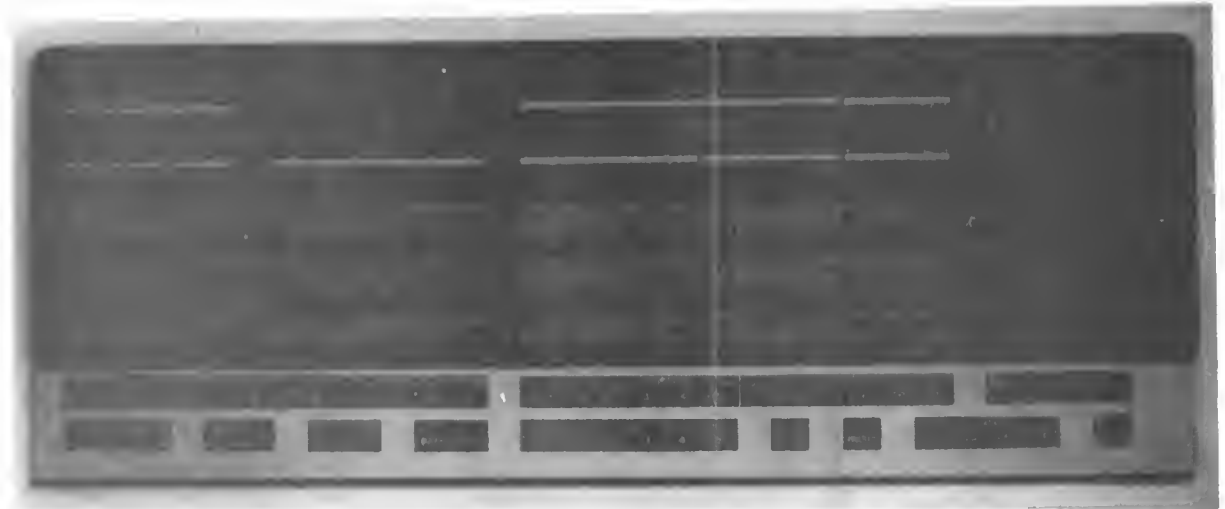


Photo 8-1. DEC front panel.

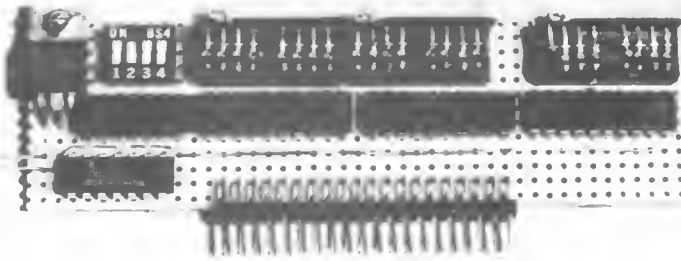
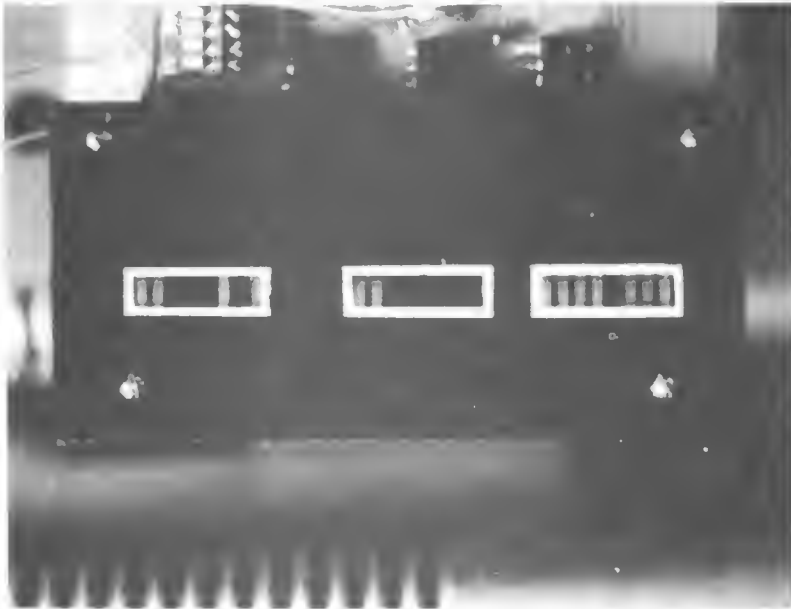


Photo 8-2. Micro front panel.

Micro front panel monitor shows data and address lines from read, write, input, or output conditions.



Micro front panel can be plugged directly into the edge card or put at the end of a connector cable, as shown here.

When using the micro front panel, you'll notice that much of the activity, particularly memory reads, run by very fast. It can't be helped. Because the TRS-80 uses dynamic memory which must be refreshed every few thousandths of a second (see description earlier in this chapter), the instruction clock cannot be stopped. Thus, you'll have to get used to the fast operation, noting from the intensity of the LEDs the frequency with which an area of memory is accessed. The monitor is remarkably useful when running diagnostic routines, because it points out whether there are any obvious signal line flaws. If the diagnostic program is a tight loop (most are – see chapter 10), then the activities of the computer's signal lines will become very obvious on the micro front panel.

In any case, it's easy to build and nifty to watch – very instructive to see how the computer accesses disks, for example, or how one data line pulses during cassette input/output.

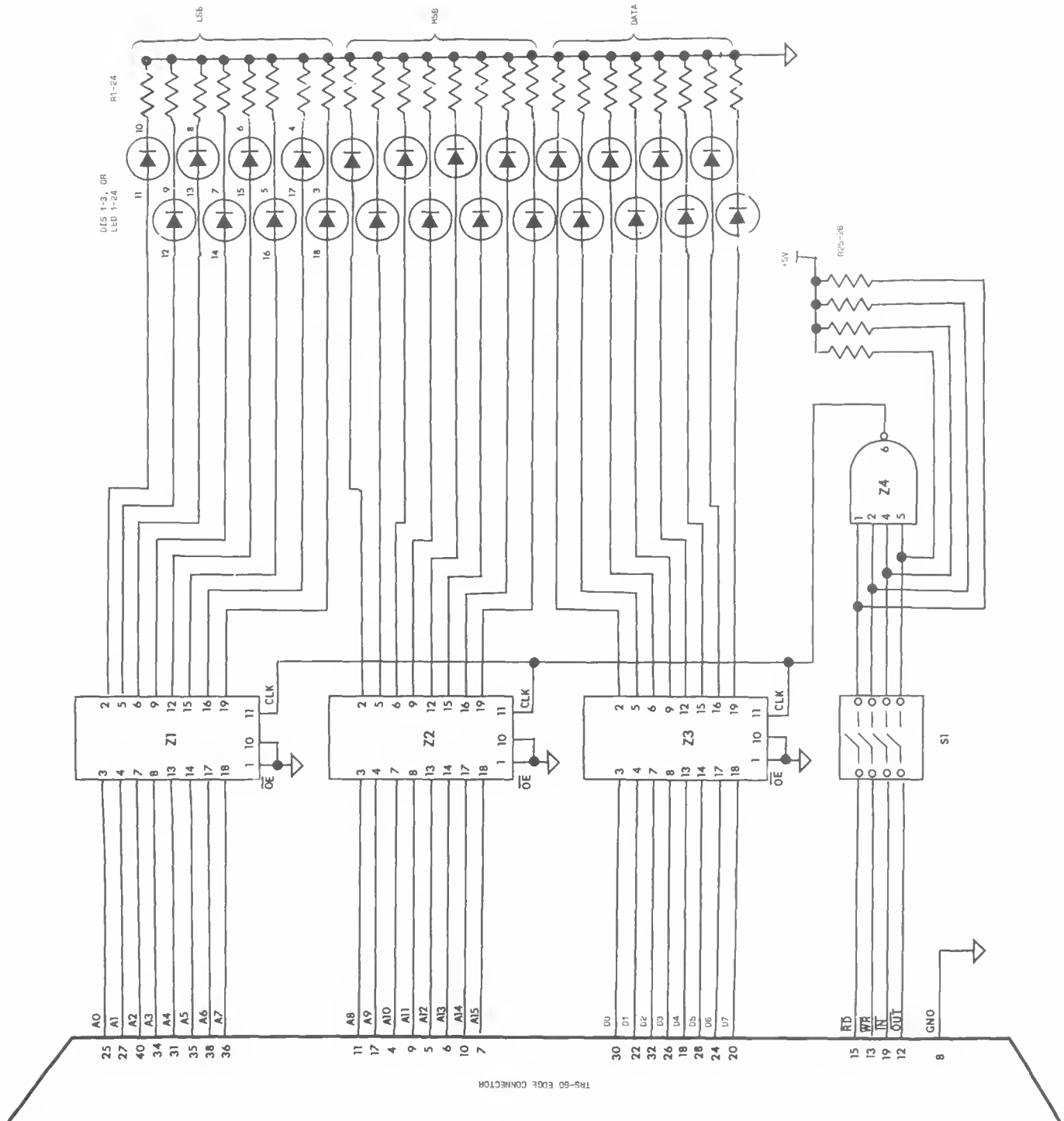


Figure 8-18. Micro front panel.

# NOTES



## Keeping It Safe: Mass Storage

Mass storage for computer programs and data has evolved from the days of punched paper tape and punch cards to present systems involving magnetic disk storage and bubble memories.

For microcomputer users, mass storage presents a unique problem: slow and apparently unreliable tape systems are inexpensive but frustrating, whereas fast disk-based systems seem transparent, trouble free, and expensive. Both of these generalizations border on myth.

What are the true advantages and disadvantages of the competing systems? This chapter will cover them in some detail, but briefly, here they are:

1. Cassette tape is a slow to medium speed storage medium (500 bits per second is normal for the TRS-80, with systems capable of transferring data at more than 3000 bits per second).
2. Tape is an inexpensive storage medium (3/1000 of a cent per bit), and is available across the counter. The hardware is pre-configured on the TRS-80, and the operating system (such as it is) is in ROM. Total system cost, including several boxes of blank cassettes, is under \$100.
3. Tape is a reliable storage medium, where good materials are used and care is taken. In a properly aligned system using low-noise, high-output cassettes, my own tests showed an average of one loading failure in 2.5 million bits.
4. Higher speed tape loaders add to the cost of the tape electronics, reducing their attractiveness, but increasing their speed and often their reliability where marginally recorded tapes are present.
5. Tape storage is sequential access, except where special digital tape systems are used. A program, once in the process of being loaded, is not tried again and again should a bit failure occur. Data storage problems are compounded, with double-dumping almost a requirement, and sequential conception and operation of programs essential.
6. The tape operating system is an essential part of the language, except where outboard devices (such as *Fastload* and similar systems) are used. Thus, it cannot be changed without adding a non-transparent, RAM-resident patch to the operating system.
7. Disk systems are capable of fast transfer rates (125,000 bits per second is possible), but the orientation towards records, directories, and other internal checking and referencing systems creates an effective transfer rate of 10,000 bits per second or less.
8. Disk systems contain built-in error checking and re-try routines. Thus, although data transfer errors occur as frequently as errors on cassette tape (if not more so), the random-access and retry capabilities make them appear error free.

except where disks have been physically damaged, written over in error, or demagnetized. The apparent reliability is very high.

9. Disk systems are comparatively costly in their original hardware configuration, their operating systems, and their medium. However, the storage cost of the medium is comparable to that of cassette tape, at about 2/1000 of a cent per bit. The initial hardware investment is higher, requiring disk control (\$300 for an expansion box or similar controller) and the drive (\$350 or more, with quality increasing with price), and an operating system (\$25 to \$250, again depending on needs). An initial investment in disk storage, including a box of disks, can begin at \$750 and end in the multiple thousands.

10. System flexibility is increased greatly, as the disk's operating system and BASIC language additions overlay each other as needed, and appear almost transparent to system operation. However, the plethora of disk operating systems and approaches limits the interchangeability of information from one TRS-80 to another with a different operating system.

11. Intermediate and hybrid systems are available that encompass some of the features of both standard tapes and disks. Foremost among these is the Exatron Stringy-Floppy endless-loop tape cartridge system. Its operating system is ROM-resident, its transfer rate is 7,200 bits per second.

12. The Stringy-Floppy is probably the most reliable mass storage system under adverse environmental conditions, putting it above tape and far beyond the sensitive (some say temperamental) disk systems. This aspect more than any other probably justifies its consideration as serious mass storage. Based on hi-tech, laboratory models, the ESF is a scaled-down scientific storage system.

13. Like tape, access using the ESF is sequential, but the endless loop makes pseudo-random access possible. With short tapes and programs less than 8K bytes, actual load/save time is faster than disk.

14. Cost of this system is less than disk (\$250 for the hardware, \$3 for the medium,

an endless-loop 'wafer'), and the cost of storage is less than all other complete systems (about 1/1000 of a cent per bit). To be competitive with dropping disk prices, I expect to see the hardware cost drop.

15. Although less fragile than disks (and higher in quality of the magnetic surface), the ESF wafers, because they use thin tape on a tiny, endless-loop hub, can be damaged by the tape binding or pulling from the housing. Unlike the larger cassettes, the tape cannot be successfully reinstalled in the housing, and unlike disks, the undamaged material on the wafer cannot be recovered.

This chapter will present a tour through the available mass storage systems (except tape – see Supplements to Chapters 3(?) and 6(?)), describe the construction of a paper tape reader, and present the construction and operation of a tape storage device using 8-track cartridges.

## Disk Drives

Disk drives come in a variety of sizes, shapes and formats. Among those are floppy disks in 5-inch and 8-inch sizes; removable and permanent hard disks; and permanently housed Winchester drives. Miscellaneous variants of all kinds are for sale or under development.

The most popular system for the TRS-80 is the 5-inch floppy disk system. The drive contains a platter which spins the magnetic disk inside its cardboard sleeve, a record/playback head to read the data, and a stepping motor to move the head from concentric track to track. The data is recorded using sharp digital pulses without any sort of audio recording considerations such as high-frequency bias. DC 'trim erase' is used to remove previously recorded data just ahead of the write head.

A single indexing hole near the center hub is used to inform the drive electronics and control software of the disk's position inside the paper sleeve. Other than this hole and a write-protect notch in the top edge of the paper case, there is no other information available to the drive and control software from a blank disk. It must be formatted, which is the process of embedding magnetic information on the disk for later use by the disk operating system.

The information needed for the original Radio Shack specifications included the magnetic outlining of 35 concentric tracks on the disk, and the separation of those 35 tracks into ten discontinuous bands each, called sectors.

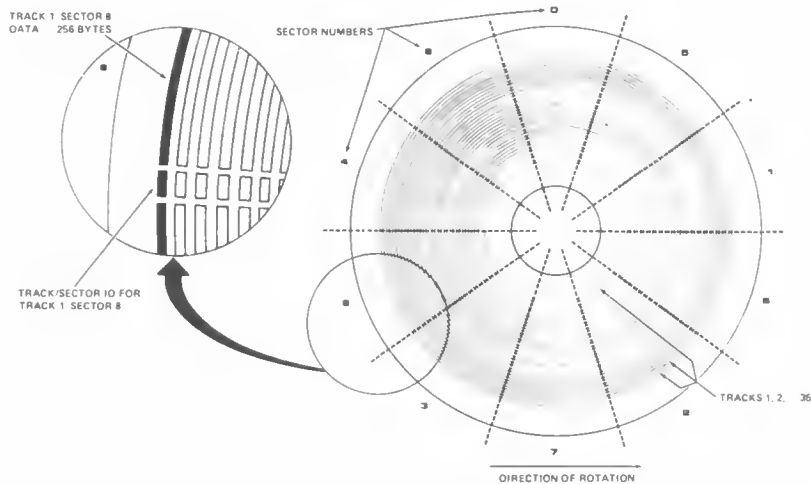


Figure 9-1. Disk and disk system format.

Details of the system by which information is recorded on the disk can be found in the *TRSDOS 9 Disk BASIC Reference Manual*, and *TRS-80 Disk 9 Other Mysteries*. In summary, the original Radio Shack specifications called for a disk containing 35 tracks of 10 sectors of 256 bytes each, for a total of 83,060 available bytes (89,600 are actually recorded, but several thousand are reserved for directory and other information – again, see the references above for details).

Drive manufacturers and software authors immediately began modifying the Radio Shack standards. 40, 77 and 80 tracks could be used, and nearly a dozen different disk operating systems (DOS) made their appearance: the original TRSDOS from Radio Shack (versions through 2.3 now issued); NEWDOS, an updated and corrected version of TRSDOS, sold by Apparat; NEWDOS/80, a complete re-write incompatible with the others; VTOS (now issued through versions 4.0), written by the original author of TRSDOS; MicroDOS and its successor OS/80, a stripped-down, efficient, minimal DOS which is growing larger and hence less attractive; DBLDOS, a Percom entry that operates its double-density (80-track) hardware option; DOSPLUS in single- and double-density versions, a wide-ranging program by Micro Systems Software; CP/M, supposedly a standard DOS for microcomputers, but available only in modified form for the TRS-80; and many special operating systems to load and run protected software.

Choosing a disk operating system is outside the range of this book, to be sure. Incompatibilities ride rampant over the bytes of the TRS-80 computer; the consistent, convenient, accessible Level II ROM gives way under disk control to the

whims of software authors and entrepreneurs, good and bad. It might seem like I'm knocking DOSes; I'm not, because it's truly a customizer's dream. But a problem arises when trying to deal with customization: to begin with, a disk operating system is a kind of customization. Hence, it becomes almost impossible to customize one further without being forced to provide versions for every popular DOS. There are too many, and they change quickly . . . to patch them invites problems, ill will, and frustration.

On the other hand, a disk system is ideal for customizing the TRS-80, because, once you have selected the operating system, you may modify and change it, making it your own. In a sense, that is how Apparat created NEWDOS – as a series of patches and improvements to TRSDOS, and in fact it originally required that disk users already own a copy of TRSDOS.

## Inside A Disk Drive

The insides of a disk drive seem incredibly simple. In fact, they are. It is only the precision of alignment needed and a few expensive parts which bring the cost so high. The drive consists of an electronic control board which is capable of communicating with the controlling computer, in this case the TRS-80, through a disk controller chip (FDC – see below). There is a motor which spins the disk, and in most inexpensive drives this motor is connected through a drive belt to the disk hub. The motor speed of 300 rpm must be accurate to within five percent; three percent deviation from normal is reasonable, and most drives are capable of a 1.5 percent long-term deviation. The disk is inserted in the housing and held in place by a cone and pressure plate which fit around the hub, and clamp gently but firmly to the disk's center area.

When the door to the drive is closed, the pressure plate moves into place, and the disk is brought into contact with the read/write head. A properly seated disk will present the indexing opening to a light sensor, so that as it spins inside the paper envelope, the index hole will open the light beam as it passes by.

A stepping motor is capable of moving precisely to one position on its axis, on command. This type of motor is used to position a magnetic read/write head on the disk track to be read. The motor is fast and precise, and the assembly which it moves is carefully machined so there is virtually no up-down play in the head. Opposite the head is a pressure pad which forces the disk

to maintain contact with the head, albeit separated from the head by a few microinches.

The head itself is usually of the glass-ferrite variety; it has an extremely smooth, highly polished surface that will not damage the disk, and a very long life that exceeds 20,000 hours of continuous head-to-disk contact. It is capable of handling the high write currents generated by the digital circuitry, and virtually immune to electronic noise in its vicinity.

In the TRS-80 system, the disk drive itself accepts and sends certain pieces of information. They are:

1. A 4-bit drive select indicator sent to the drive; only the drive hard-wired to accept this signal will respond. In the TRS-80 system, this wiring is done in the cable itself.
2. A motor-on signal, which turns on the 300 rpm hub motor.
3. A track-to-track stepping signal and a stepping direction signal.
4. A write-enable signal and a stream of written data.

5. A stream of written data sent by the drive.

6. A write-protect signal to prevent writing to write-protected disks. Most drives will not respond to a write signal, and this write-protect signal is sent so the software can report a write-protected condition.

7. An index pulse to indicate where the disk is currently located in its rotation.

8. A track-zero indicator to identify when the disk head has reached the outermost track on the disk. This is used to locate tracks, relocate tracks or reposition the head, and on initial access to identify the head position.

Except for the drive-select signals, the disk controller chip is responsible for managing all these lines. In the TRS-80, a type 1771 controller is used, manufactured by *Western Digital*. A complete data sheet is provided with the Expansion Interface service manual from Radio Shack.



*The author at work.*

This controller has eight data input/output lines (DAL0 to DAL7) to the computer, an interrupt output to signal its need for service (INTRQ), read and write enables (RE and WE), and a two-bit address select (A0 and A1). These last are used to select the register that will send or receive data on the DAL lines:

A1	A0	RE	WE
0	0	Status Register	Command Register
0	1	Track Register	Track Register
1	0	Sector Register	Sector Register
1	1	Byte Register	Data Register

.....

In the TRS-80, the chip is always selected (CS) because the buffering of data is taken care of by Z33, Z37 and Z38 in the expansion box. The Data Request signal (DRQ) is not used, nor are the three-phase motor signals (PH3 and 3PM), and the track-greater-than-43 signal (TG43). These signals would be present were a more sophisticated disk drive capability intended by Radio Shack in future versions of the TRS-80.

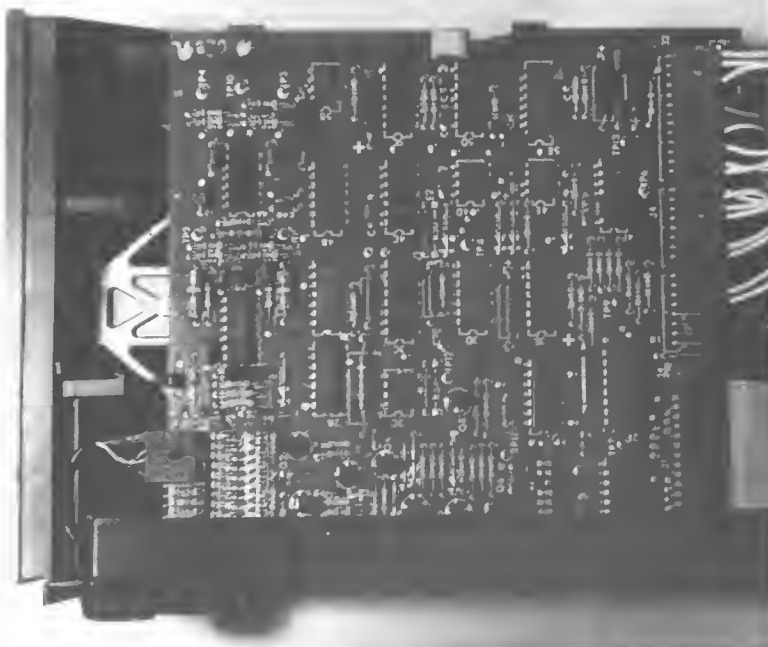
The clock input is provided by the separate oscillator in the expansion interface. Three voltages are needed (+12, +5, and -5), plus ground.

Test, disk initialization (DINT), and write-fault lines (WF) are not used, nor are the infamous external data separation (XTDS) and external data clock (FD CLOCK) lines. Disk data are separated by clock pulses, and high accuracy demands unfailing differentiation between the clock pulses and the data pulses. The Percom data separator plugs into the controller chip socket and makes use of XTDS and FD CLOCK.

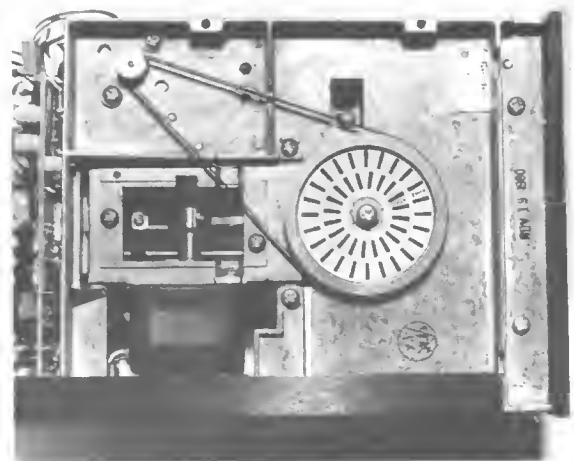
The head-load output (which determines whether the read/write head is in place against the disk) is not used, because software controls the timing between motion of the read-write head. Head-load timing (HLT) and READY are both connected via external logic to the TRS-80, where software determines when the drive and read/write head should be ready to read or write. The remaining disk controller signals lead to and from the disk drive itself, and are identical to those listed in the description of the drive signals.

Photo 9-1. Pictorial tour (6) of disk drive.

Front door of disk drive controls mechanism to steady the disk as well as a switch to indicate that the door is closed.

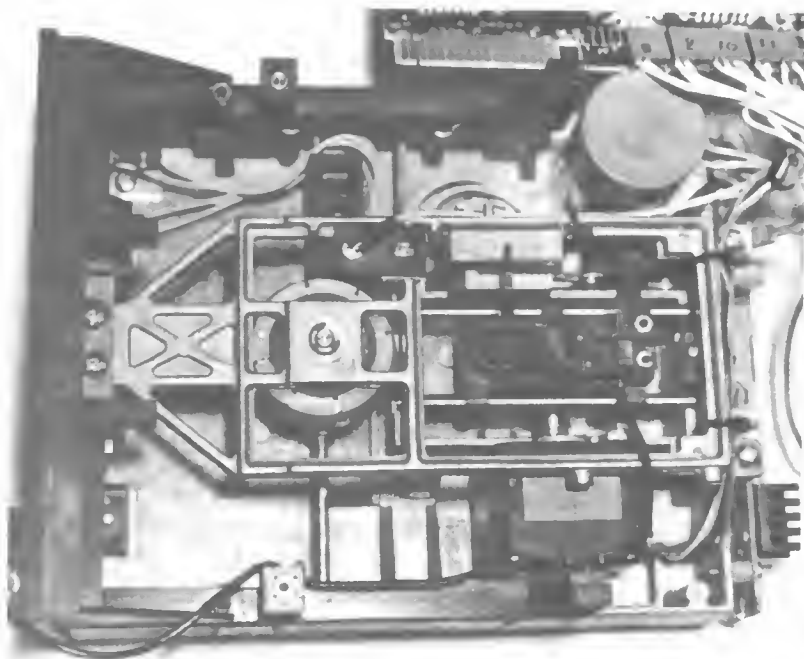


Circuit card contains disk read/write circuitry, motor stepping control, and computer interfacing. Computer connector and termination resistors are seen at lower right.



Opposite side from control card is the drive motor with speed strobe disk, and a window revealing the shaft of the head stepping motor.

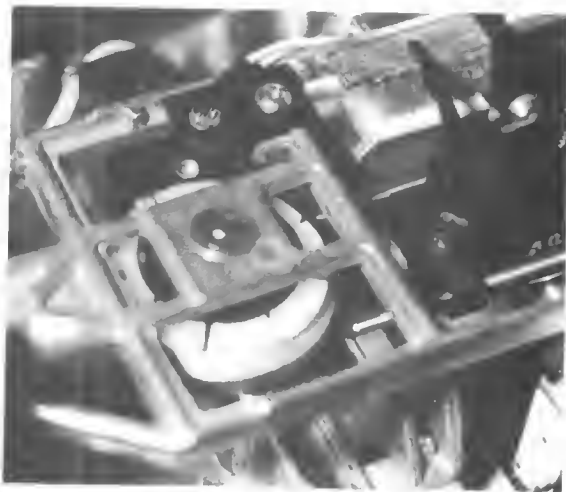
## Inside a Disk Drive



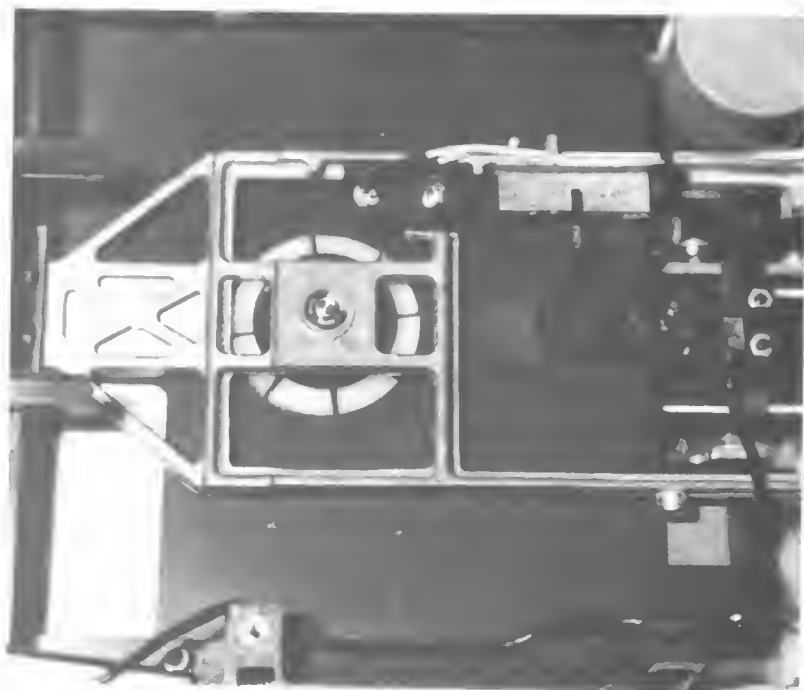
*Removing the control circuit card reveals a heavy cast frame to hold the disk in place. In center is the cone that fits through the disk; to the right is a pressure pad.*



*Close-up of read/write head and pressure pad. White square is glass ferrite head surface, vertical stripe is the head's read/write surface. Pressure pad moves into place when the disc's front door is closed.*



*Close-up of positioning sensor. Index hole of a disk allows light to pass from LED emitter to sensor (top center).*



*The drive with a disk in place. Cone fits through disk and clamps it in place. Head and index hole light emitter are on the other side of the disc; pressure pad and light sensor are on this side.*



*Hefty disk drive power supply handles motor drive current and powers the electronics.*

## A Heavy Dose of DOSes

On the software side, the number of disk operating systems for the Model I continues to increase. Most include 'Level III BASIC', the usual copying and formatting capabilities, but have been expanded to include more than that. For a complete description of the features of competing DOSes, refer to the March 1981 issue of 80 Microcomputing. Among the special features of the most prominent DOSes:

**TRSDOS.** AUTO, ATTRIB, CLOCK, COPY, DATE, DEVICE, DIR, DUMP, KILL, FREE, LIB, LIST, LOAD, PRINT, PROT, RENAME, TIME, VERIFY. Also BASIC, BASIC2, DEBUG, TRACE. Level III BASIC functions. Details are found in the TRSDOS

manual, and other DOSes include all of these commands in some way or other.

**NEWDOS+.** Adds COPY, JKL screen print, DIRCHECK directory verification, LMOFFSET tape load offset module, EDTASM with modifications, SUPERZAP for modifying disk contents, LEVEL 1 located in RAM, DISASSEM, LV1DSKSL disk save/load for Level I. CMD"DOS COMMAND NAME" to execute from BASIC, and several commands to re-enter BASIC from DOS.

**NEWDOS/80.** CHAIN, HIMEM protection from DOS, JKL, MINIDOS don't-disturb-memory DOS, MDBORT for killing MINIDOS, PDRIVE setup for multiple drive types, PURGE for killing file groups, SYSTEM to create special commands, SUPERZAP, LMOFFSET, LEVEL 1, ASPOOL print spooler, DIRCHECK.

**VTOS.** BOOT software reset, BUILD a group of auto-execute programs, CHAIN to execute them, MEMORY, PURGE, RUN for non-VTOS systems, SYSTEM, XFER disk copy program, PATCH disk modification routine, VTCOMM communications utility, KSR terminal program, ROUTE for changing the destination of data, SET for a user device program, SPOOL, RESET to cancel device setup, LINK for devices, FILTER to be used with device routing, ALLOC setting up disk space in advance.

**DOSPLUS.** Adds BOOT, BUILD, CLEAR a directory file, DO group of auto-execute programs, FORMS to set up the printer driver, a different variant of FREE, PAUSE for user input in auto-execute routines, RS232 for a report on that status, PURGE, DISKZAP for modifying disk information, CLRFILE for zeroing a file, COPY1 copying utility, CRUNCH space compression for BASIC, TRANSFER program copier.

**ULTRADOS.** CLEAR for zeroing memory, DEAD for zeroing memory, TOPMEM for setting DOS-protected memory, CMD"C" compression routine, CMD"DOS COMMAND NAME" which executes from BASIC, CMD"O" file buffer allocation, CMD"X" to return to BASIC from DOS, cross referenced listing of variables and line numbers, renumbering from BASIC, shorthand command keystrokes.

## A Garden Full of Varieties

The available variety of disk drives is growing. Not only are the capabilities of standard 5-inch drives being stretched (to run 40, 77 and 80 track densities), but 8-inch systems and hard-disk systems are being introduced for the lowly TRS-80. Among them:

Manufacturer	Model/Type	Price	Comments
Minifloppy (5-inch) Disc Systems			
Access	AFD-100	\$315	40 tracks; Tandon?
Aerocomp	40-1	\$350	40 tracks; double-density
	80-1	\$460	80 tracks; double-density
	80-2	\$460	40 tracks; double-sided; double density.
	160-2	\$600	80 tracks; double-sided; double density.
CPU Shop	CCI-100	\$315	40 tracks; Percom?
	CCI-200	\$430	80 tracks; Percom?
MPI	B/51	\$320	40 tracks; 5 ms track-to-track; auto-eject; double-density head.
	B/91	\$425	80 tracks; 3 ms track-to-track; auto-eject; double-density head.
Micropolis	MCP1027	\$300	35 tracks, single head
	MCP1037	\$700	35 tracks, dual head
	MCP1027-2	\$440	77 tracks, single head
	MCP1037-2	\$800	77 tracks, dual head
Percom	TFD-100	\$350+	40 tracks
	TFD-200	\$650+	77 tracks
Pertec	F0200	\$380	40 tracks
Shugert	SA-400	\$330	35 tracks
	SA-410	\$340	40 tracks
Siemens	F00 100-5	\$275	40 tracks, double density
Tandon	—	\$...	40 tracks
TEAC	—	\$300	40 tracks
	—	\$400	80 tracks
Vista	V-80	\$400	40 tracks
	V-800	\$600	80 tracks
	V-8800	\$775	160 tracks

## The Exatron Stringy-Floppy

In between tape systems and disk systems is an unusual device. I only say 'in between' because the name Stringy-Floppy implies that it somehow is a tapelike version of a floppy disk device. In fact, it is not, for two major reasons: it is a unique, high reliability, high endurance storage medium; and it does not (at least in its TRS-80 configuration) contain formatting, sectoring, or record-keeping of the type used for disks.

The *Exatron Stringy-Floppy* (ESF) consists of a small DC motor which uses a tiny plastic belt to drive a capstan, exactly as in a cassette player. The motor's speed, however, is set to 10.5 inches per second, quite a bit faster than cassette, 8-track, or even open-reel. Thus, even though the tape itself is only 1/16 of an inch wide, reasonable data recording can be expected.

In the TRS-80 version, the ESF does not use any standard method of recording. A separate sequence is used to prepare, or 'verify' an endless-loop tape wafer than that which is used to record data. Both methods use a variety of bi-phase recording, in which the bit being read or recorded depends on the polarity of the bit just written or read.



Photo 9-2. Pictorial tour (6) of ESF system.

*Exatron Stringy-Floppy placed next to Microconnection modem. Both are compact, light devices.*

This makes the ESF reliable in spite of a motor speed which varies more than ten percent in either direction of its ideal speed. It also means the ESF can work under harsh conditions which might otherwise throw a standard clock/data system far off.

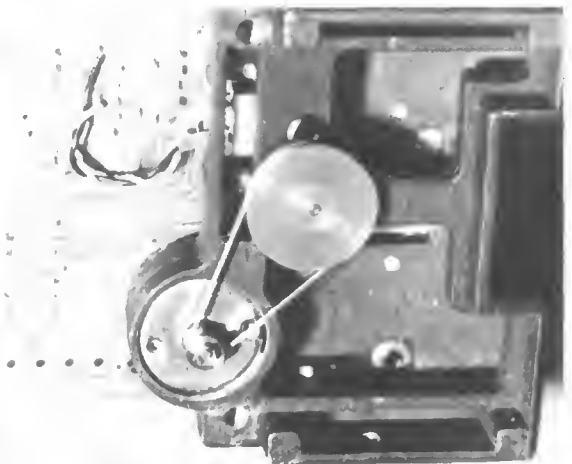
Programs are recorded on the ESF in a continuous stream, preceded by a leader and followed by information to assist the software in locating the next available blank program space. Programs and data may be read in any order, but must be written in ascending numerical order.

The ESF operating system is contained in a read only memory which resides in the unassigned memory area on the Model I (3000 to 37C0); thus, it is always available and relatively crash-free. It does use a few bytes of RAM, however, and must also patch into the Level II BASIC parameters in low memory.

The wafer itself is available only from *Exatron*, and is based on the wafers used in industrial applications. It consists of a length of high-output digital recording tape, highly polished. This is wound on a small hub, lubricated, and spliced into an endless-loop. A reflective splice is used to determine end-of-tape (EOT/BOT), and is viewed through a window at the top of the wafer.

The wafers may be write-protected by affixing a reflective-tape dot on the wafer. This is read by the operating system before the drive is activated. The ESF operating system consists of these commands:

**@LOADx** Load a program; x is the program number, and is optional. An error in the load



*ESF 'rubber band' drive is not as reliable as disk drives, but software is capable of handling greater variations in speed. Small DC motor replaced by a direct-drive type in later versions, say Exatron officials.*

terminates the process and the error (parity or checksum) is reported to the user.

**@SAVE $x$**  Saves a BASIC program; the  $x$  is the program number, and is required. The program is verified, and any saving errors are reported.

**@SAVE $x,r,s,t$**  Saves a block of memory;  $x$  is the program number,  $r$  is the starting address, and  $s$  is the length of the block. These are required. The  $t$  is an optional entry address. All numbers are in decimal. The block is verified, and any saving errors are reported.

**@NEW $x$**  Verifies a wafer. When  $x$  is used, the wafer is cleared and verified from that program to the end splice. Available byte count is reported.

## Fastload, TC-8, and Other Systems

Most other alternatives to disk systems involve using standard cassettes for high-speed saving and loading. Among these are Fastload, manufactured by *Personal Microcomputers, Inc.*; TC-8, or the 'Poor Man's Floppy', made by *JPC Systems*; and the Beta-80, made by *Meca Technology*.

The advantage to the standard cassette is simple: cost and availability. Unlike disks, they are able to take some measure of abuse, and can be replaced without regret over the expense. And

unlike Exatron wafers (or disks, for that matter), they can be purchased if necessary in the local grocery. Furthermore, cassette tape technology has progressed further than disk technology, providing better surfaces, adhesion, and signal-to-noise ratio. Disks are still in the dark ages of reproduction compared to audio tape.

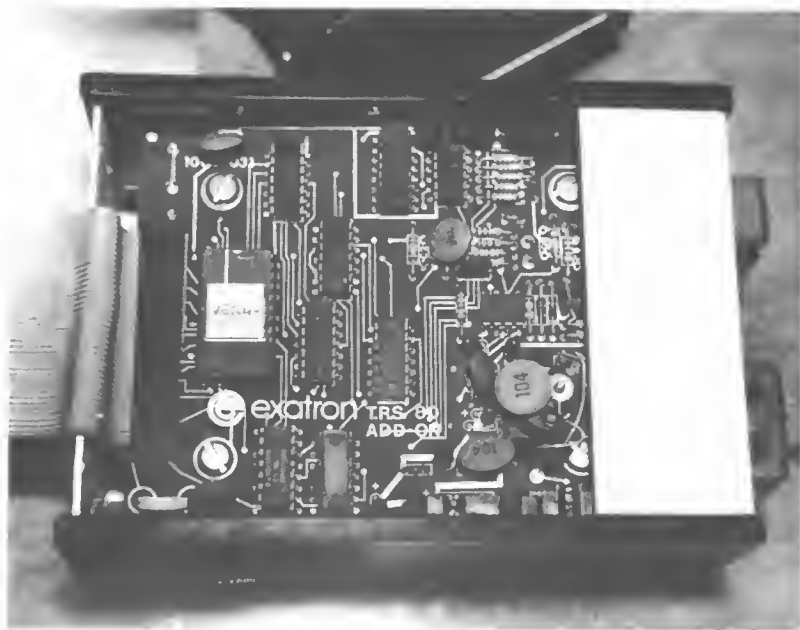
Fastload is a ROM-based operating system combined with a hardware detection and shaping circuit. A modified CTR-41 is used, where the fast-forward button and play button can be locked down. The circuit then can read a standard 500-baud tape in the fast-forward mode quite reliably at about 8000 baud. Debounce, audible beep, and key repeat are included with the operating system.

Although I have not examined the schematics for Fastload, its carefully designed circuit board, with voltage regulator heat-sinked to the case, attest to a cautious, probably over-designed system. The loading system is put into operation by typing SYSTEM (ENTER), /12288 (ENTER); optional debounce/beep/repeat can be added as well.

When in operation, Fastload uses the single LOAD command in Level II, and is compatible with disk-based computers by using a SYSTEM call. The 500-baud load and save are left undisturbed. Fastload is well-designed, and reasonably reliable. If your original tapes have audible 'bumps' when played at high speed, however, they may not load easily with this system. Commercial tapes, or those recorded by the user on virgin cassettes, load easily and extremely fast. 'Bumpy' tapes, however, do not load well, and have to be re-recorded.

One of the other disturbing features of Fastload is its tendency to give a 'READY' message even when a BASIC program has been loaded incorrectly. Granted, it is almost impossible to be sure a BASIC program has loaded well because, in its normal CSAVE format, no checksums have been provided. But the user should be sure to list the program before expecting it to run. SYSTEM loads, on the other hand, present checksum error messages on bad loads. Fastload costs \$188 assembled; a modified CTR-41 is \$95.

TC-8, or the Poor Man's Floppy, both saves and records at high speed. It is provided in kit form; assembly can be completed in a few hours. Like Fastload, the TC-8 plugs into the edge connector, but the software is RAM-resident. The standard CTR-80 tape recorder is used, and a two-wire swap is shown to make the CTR-41



Control card of ESF contains drive electronics, tape read/write circuitry, and 2716 EPROM containing the ESF operating system. Power supply is underneath the card.

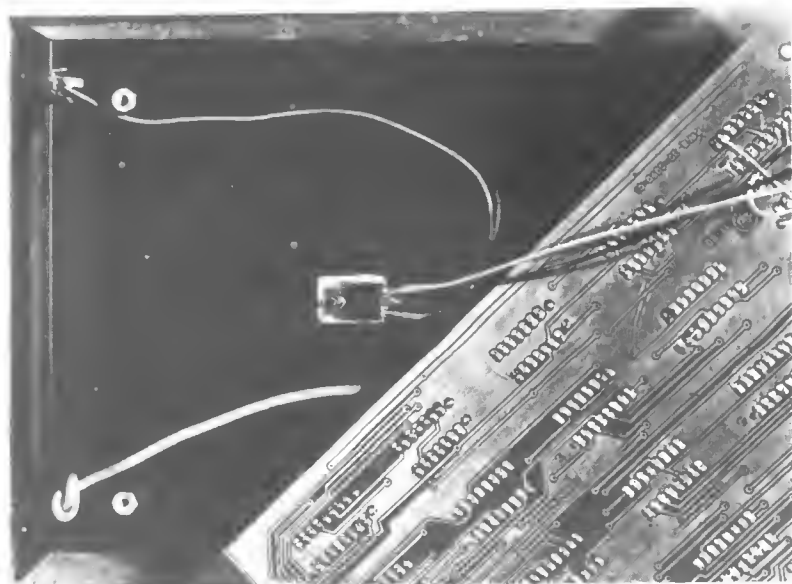


Photo 9-3. Fastload device.

Instead of a small heat sink, Fastload power supply regulator is sunk to the entire metal case. Insulators around regulator show attention to detail.

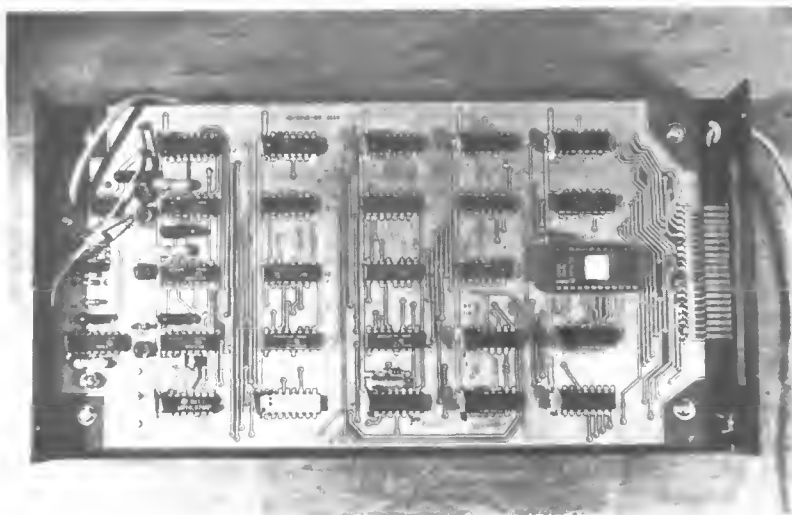


Photo 9-4. Fastload circuit card.

Circuit card for Fastload shows extremely conservative design, with careful attention to grounding and possible noise sources. DIP resistor package is used in place of separate resistors, and the operating system is contained in a 2716 EPROM. Since less than 1K of the EPROM is used for the operating system, the remainder may be programmed by the user with utilities.

run properly (its high current tended to fuse motor relays together on the TRS-80).

Since it is both a record and save device, the TC-8 contains fifteen separate commands, including SAVE (BASIC programs only) and PUT (machine language blocks); LOAD BASIC programs next or by name; LOAD? for verification; LOADN to position the tape; comparable GET commands for machine language programs; RUN as a load-and-go command; RSET for cassette motor on; OPEN, CLOSE, PRINT, and INPUT for file management; and KILL to eliminate file management.

The software can be reconfigured, relocated, and stripped to a memory-saving bootstrap version. The TC-8 is provided with extensive documentation which is clear and literate, providing examples, a sample program using the OPEN, PRINT#, INPUT#, and CLOSE commands, along with recommendations, suggestions, and warnings (plus sympathy!) about the problems of transferring some commercial machine-language software to the TC-8.

The circuit is quite simple, but well designed and accurate; the capabilities of this elegant circuit comes from the software. The construction manual provides soldering suggestions and explanations, recommendations on the soldering iron to purchase, and drawings of properly soldered connections. For convenience, all work is done on the underside of the board; correct placement and orientation of parts is emphasized. Two-color layouts of the board are provided on every page, along with a checklist of each step. These are some of the finest assembly instructions I have seen for a project.

When complete, the unit loads and saves at 2500 baud on off-the-shelf cassettes. The company recommends certain brands, and also sells the brand JPC products uses. All signals to the tape player are non-critical, and the loading routine will accept signals over most of the audible, undistorted output spectrum of most cassette recorders. TC-8 costs \$90 in kit form.

The Beta-80 is a tape storage system patterned after professional digital tape devices. Although a standard digital cassette is used, the Beta-80 is configured something like a disk, with directory and fast access. Fast forward and rewind are automatically controlled by the RAM-resident operating system, and the search speed is 100 inches per second.



Photo 9-5. TC-8 tape system.

TC-8 tape system attaches to edge-card connector and cassette recorder to provide high-speed record/save of programs. Photo courtesy JPC Products Company.

```

00100 ; #####
00110 ;
00120 ; HIGH SPEED TAPE LOAD AND SAVE ROUTINE OPERATING AT 1960
00130 ; BAUD. THIS CAN BE MADE TRANSPARENT TO BASIC BY USING
00140 ; EITHER /SAVE AND /LOAD OR /PUT AND /GET. THE ROUTINE
00150 ; PRESENTED HERE SAVES A BLOCK OF MACHINE CODE, BUT MAY
00160 ; BE USED FOR SAVING AND LOADING BASIC BY LOCATING START
00170 ; AND END OF THE BASIC PROGRAM (FOR WRITE), AND START AND
00180 ; END OF BASIC MEMORY (FOR READ). SAMPLE ACCESS ROUTINES
00190 ; ARE PRESENTED AT THE END OF THIS LISTING.
00200 ; #####
00210 ;
00220 ; ORG 7000H ; TOP OF MEMORY - RELOC.
00230 ;
00240 ; #####
00250 ; WRITE-BYTE SUBROUTINE. BYTE IS ASSUMED IN C REGISTER
00260 ; #####
00270 ;
7000 060B 00280 WRITE LO B,B ; NUMBER OF BITS TO WRITE
7002 C5 00290 PUSH BC ; SAVE BITS OF BYTE
7003 0620 00300 LO B,20H ; GET A DELAY VALUE
7005 10FE 00310 OJNZ $ ; AND DELAY 117 USECS
7007 C1 00320 POP BC ; AND RESTORE BITS
700B C01470 00330 WLOOP CALL TIMEX ; WRITE TIMING BIT
700B CB11 00340 RL ; ROTATE BIT INTO FLAG
7000 17 00350 RLA ; ROTATE FLAG INTO A
700E 0C3670 00360 CALL BITEX ; WRITE BIT TO TAPE
7011 10F5 00370 OJNZ WLOOP ; DO IT TOTAL OF 8 TIMES
7013 C9 00380 RET ; BACK FROM WRITE ROUTINE
00390 ;
00400 ; #####
00410 ; ROUTINE TO WRITE TIMING BIT POSITIVE-NEGATIVE TO TAPE
00420 ; #####
00430 ;
7014 C5 00440 TIMEX PUSH BC ; SAVE B ON STACK
7015 3A3040 00450 LO A,(4030H) ; GET SCREEN INFORMATION
701B E6FC 00460 AND OFCH ; MASK OUT LOW TWO BITS
701A F601 00470 OR 1 ; SET LOWEST BIT
701C 03FF 00480 OUT (OFFH),A ; WRITE BIT TO TAPE
701E 0607 00490 LO B,07H ; TIMING VALUE @ 1.77 MHZ
7020 10FE 00500 OJNZ $ ; IS EQUAL TO 67 USECS
7022 E6FC 00510 AND OFCH ; MASK OUT LOW TWO BITS
7024 F602 00520 OR 2 ; SET NEXT LOWEST BIT
7026 03FF 00530 OUT (OFFH),A ; WRITE NEG. BIT TO TAPE
702B 0607 00540 LO B,07H ; TIMING VALUE @ 1.77 MHZ
702A 10FE 00550 OJNZ $ ; IS EQUAL TO 64 USECS
702C E6FC 00560 AND OFCH ; MASK OUT LOW TWO BITS
702E 03FF 00570 OUT (OFFH),A ; WRITE NEUTRAL BIT

```

Listing 9-2. High-speed tape loading routine.

Loading is at 4000 baud (tape running at 5 i.p.s.), twice the speed of the TC-8, but about half that of the Exatron Stringy-Floppy and Fastload, but access time (using the directory) is less than a minute for more than a half megabyte of stored programs.

Commands are LOAD in the form of load, load and run, and load array information; SAVE for programs and arrays; MERGE for append (not a true merge) or append and run; and KILL to delete a program or an array. Other commands operate within the various RAM-resident systems.

The Beta-80 uses a reliable Phi-deck drive, and has impressive features. However, as of this writing, users report little support from Meca for converting and saving machine language programs on the Beta-80. The Beta-80 costs under \$300.

## High-Speed Cassette Loading

Speeds higher than 500 baud are achievable entirely through software, and most systems work exceedingly well. Among the most popular are: SPEED, the first such program, loading and saving at 1500 (?) baud; HISPED (Palomar Software, 170 S. Palomar Dr., Redwood City, CA 94062; \$24.95), a 2000-baud system; ZIPLOAD, in the public domain and published in the 80 Encyclopedia (Wayne Green, Inc.); and B-17 (ABS Suppliers, P.O. Box 8297, Ann Arbor, MI 48107; \$25).

Each of these save/load programs patches into the BASIC operating system, and all use the DOS-reserved LOAD and SAVE commands. The last is of particular interest because it is a complete system rather than merely a save/load program.

SAVE, using a six-character name, sends BASIC programs to tape; LOAD? verifies them. LOAD puts the computer into the SYSTEM mode automatically and loads a BASIC program. PUT and GET are used for formatted file arrays. It is also provided with a machine language module. Checksum errors, full memory buffer, continuity errors, and format errors are reported by this module. The entire source code is available for sale if the B-17 program is also purchased.

Aside from B-17's high reliability, it also provides an on-screen prompt indicating saving and loading.

What makes a high-speed loader not only

## A Paper Tape Reader

```

7030 0607      00580      LO      B,07H      ; TIMING VALUE @ 1.77 MHZ
7032 10FE      00580      OJNZ     #          ; IS EQUAL TO 113 USECS
7034 C1        00600      POP      BC          ; RESTORE NUMBER OF BITS
7035 C9        00610      RET          ; BACK TO BIT WRITE ROUT.
              00620      ;
              00630      ; #####
              00640      ; ROUTINE TO WRITE INDIVIDUAL BIT TO TAPE (OR NOT IF 0)
              00650      ; #####
              00660      ;
7038 C5        00670      BITEX    PUSH    BC          ; SAVE NUMBER OF BITS
7037 E601      00680      AND      1          ; MASK OUT ALL BUT BIT 0
7038 47        00680      LO      B,A        ; SAVE A IN B REGISTER
703A 3A3040    00700      LO      A,{4030H}    ; GET SCREEN STATUS
7030 E6FC      00710      AND      0FCH      ; MASK OUT LOW TWO BITS
703F B0        00720      AOR      A,B        ; SET BIT OR NOT
7040 03FF      00730      OUT      [OFFH],A    ; SEND OUT CASSETTE PORT
7042 C5        00740      PUSH    BC          ; GAVE B REGISTER AGAIN
7043 0805      00750      LO      B,05H      ; TIMING VALUE FOR BIT
7045 10FE      00760      OJNZ     #          ; DELAY IS 65 USECS
7047 C1        00770      POP      BC          ; GET VALUE BACK INTO B
7048 C800      00780      RLC      B          ; ROTATE INTO BIT 1
704A E8FC      00790      AND      0FCH      ; MASK OUT LOW BITS AGAIN
704C B0        00800      AOR      A,B        ; SET BIT 1 OR NOT
7040 03FF      00810      OUT      [OFFH],A    ; SEND OUT CASSETTE PORT
704F C5        00820      PUSH    BC          ; SAVE B REGISTER AGAIN
7060 0806      00830      LO      B,0BH      ; DELAY IS 63 USECS
7062 10FE      00840      OJNZ     #          ; DELAY FOR BOTTOM OF BIT
7054 E6FC      00850      AND      0FCH      ; CREATE A NEUTRAL BIT
7058 03FF      00860      OUT      [OFFH],A    ; AND WRITE IT TO TAPE
7068 0808      00870      LO      B,0BH      ; GET DELAY VALUE IN B
705A 10FE      00880      OJNZ     #          ; DELAY IS 110 USECS
705C C1        00880      POP      BC          ; CLEAR STACK OF BIT
7050 C1        00800      POP      BC          ; GET ORIGINAL BITS BACK
705E C9        00910      RET          ; BACK TO SAVING ROUTINE
              00920      ;
              00930      ; #####
              00940      ; ROUTINE TO WRITE LEADER OF 256 BYTES OF FF PLUS SYNC A5
              00950      ; #####
              00960      ;
705F AF        00970      SYNCHR   XOR      A          ; DEFINE DRIVE NUMBER 0
7060 C01202    00980      CALL     0212H      ; DRIVE MOTOR RUNNING
7063 010000    00990      LO      BC,0        ; NEED A LOOP ABOUT 1 SEC
7066 C06000    01000      CALL     0060H      ; CALL DELAY IN ROM
7068 C5        01010      SYLOOP   PUSH    BC          ; SAVE THIS VALUE (BC=0)
706A 0E00      01020      LO      C,0        ; WRITE BYTE OF 1111 1111
706C C00070    01030      CALL     WRITE    ; WRITE THAT BYTE
706F 0608      01040      LO      B,B        ; SHORT LOOP BETW. BYTES
7071 10FE      01050      OJNZ     #          ; DELAY JUST 58 USECS
7073 C1        01060      POP      BC          ; RESTORE VALUE FOR USE
7074 10F3      01070      OJNZ     SYLOOP    ; WRITE TOTAL OF FF BYTES
7076 0EAS      01080      LO      C,0A5H      ; SYNCHRONIZATION BYTE
7078 C00070    01090      CALL     WRITE    ; WRITE THAT BYTE
707B C8        01100      RET          ; BACK TO MAIN ROUTINE
              01110      ;
              01120      ; #####
              01130      ; READ BYTE SUBROUTINE: COMPLETED BYTE INTO C REGISTER
              01140      ; #####
              01150      ;
707C 0608      01160      READ     LO      B,B        ; NUMBER OF BITS TO READ
707E C0BA70    01170      RLOOP    CALL     REDEX     ; READ TIMING BIT
7081 C0BC70    01180      CALL     REOBIT    ; READ DATA BIT IF ANY
7084 17        01190      RLA          ; ROTATE BIT INTO CARRY
7085 C811      01200      RL      C          ; ROTATE CARRY INTO C
7087 10F5      01210      OJNZ     RLOOP    ; DO IT TOTAL OF B TIMES
7088 C8        01220      RET          ; BACK FROM READ ROUTINE
              01230      ;
              01240      ; #####
              01250      ; ROUTINE TO READ TIMING BIT FROM TAPE AND DELAY AFTER
              01260      ; #####
              01270      ;
708A C0C770    01280      REDEX    CALL     INSIG     ; KEEP LOOKING FOR ONE
7080 08FF      01290      REOX2   IN      A,[OFFH]    ; KEEP LOOKING FOR ONE
708F 17        01300      RLA          ; ROTATE INTO CARRY
7080 30FB      01310      JR      NC,REOX2    ; KEEP LOOKING FOR ONE
7082 C5        01320      PUSH    BC          ; SAVE NUMBER OF BITS
7083 080F      01330      LO      B,0FH      ; VALUE TO DELAY
7095 10FE      01340      OJNZ     #          ; AND DELAY FOR 144 USECS
7097 C4        01350      POP      BC          ; RESTORE NUMBER OF BITS
7088 C0C770    01360      CALL     INSIG     ; RESET INSIG FLIP-FLOP
7098 C8        01370      RET          ; BACK TO MAIN ROUTINE
              01380      ;
              01390      ; #####
              01400      ; ROUTINE TO READ INDIVIDUAL DATA BIT (IF ANY) FROM TAPE
              01410      ; #####
              01420      ;
709C C5        01430      REOBIT   PUSH    BC          ; SAVE VALUE IN B AND C
7090 061B      01440      LO      B,1BH      ; VALUE FOR DELAY
709F 10FE      01450      OJNZ     #          ; EQUAL TO 197 USECS
70A1 08FF      01460      IN      A,[OFFH]    ; GET VALUE FROM CASSETTE
70A3 C1        01470      POP      BC          ; RESTORE VALUES TO B & C
70A4 C8        01480      RET          ; RETURN WITH VALUE IN A
              01490      ;
              01500      ; #####
              01510      ; THIS ROUTINE READS THROUGH THE LEADER FOR SYNC BYTE
              01520      ; #####
              01530      ;
70A5 AF        01640      REOSYN   XOR      A          ; DEFINE DRIVE NUMBER 0
70A6 C01202    01650      CALL     0212H      ; DRIVE IS RUNNING
70A8 C0BA70    01660      CALL     REDEX     ; READ TIMING BIT
70AC C0BC70    01670      CALL     REOBIT    ; GET BIT FROM TAPE
70AF 17        01680      RLA          ; ROTATE INTO CARRY FLAG
70B0 30F3      01690      JR      NC,REOSYN    ; KEEP LOOKING FOR A 1

```

possible, but reliable, especially since the TRS-80's 500-baud rate seems to be full of flaws? As noted in the Supplement to Chapter 6 (?), the 500-baud rate is actually fairly reliable, except that a few misconceptions on the part of the ROM designers led to incorrect load timing. The writers of the high speed loaders noted above took considerably more care in designing their input/output schemes, and thus achieved that reliability.

Listing 9-2 presents a pair of high-speed input/output modules that load and save blocks of memory at 2000 (?) baud. These modules may supplant the CALLs to 0235 in all the input/output routines presented elsewhere in this book, allowing faster saving and loading of memory blocks, data, screens, etc. These routines use the normal cassette output. For a high-speed loader using an 8-track system and digital, very reliable recording, see Chapter 9.

## A Paper Tape Reader

One of the more common sights in the early days of larger computers was a bank of spinning tape reels for magnetic and punched tapes. Both still exist, but are not common storage modes for microcomputers like the TRS-80.

However, you may have the chance to pick up some terrific programs written for an 8080-based computer, especially those in the National Semiconductor library. These include double-precision mathematical subroutines, text handlers, light-pen readers, etc., and are sometimes available at low prices. But they are stored on rolls of paper tape. Software for devices such as the Computalker Speech Lab is also provided on paper tape.

Furthermore, though paper tape may no longer be the popular program storage medium it once was, for archival storage or communicating among different styles and types of computers, it still has its place. For occasional use, then, paper tape can be used, but an expensive reader won't be a very good investment. For less than \$50, you can interface the TPR-1 paper tape reader and the TRS-80.

The TPR-1 reader is sold by *Raeco*, Box 165, Washington, Maine 04574. The unit consists of a machined, brushed aluminum track for the paper tape, and a circuit board attached to the track. On the board are two integrated circuits, and nine light sensors are on the board under holes in the track; also provided are an LED test light, resistors, and a 14-pin DIP socket. It is sold with a good technical manual for \$32.50; an optional

```

70B2 CB11      01600      RL      C      ; ROTATE INTO C REGISTER
70B4 0607      01610      LO      B,7    ; NUHBER OF BITS LEFT
70B6 C0BA70    01620      CALL     REOEX   ; READ TIMING BIT
70B8 C09C70    01630      CALL     REOBIT  ; GET BIT FROM TAPE
70BC 17        01640      RLA         ; ROTATE INTO CARRY FLAG
70B0 CB11      01650      RL      C      ; ROTATE INTO C REGISTER
70BF 10F5      01660      OJNZ     SYNCLP  ; 00 IT 7 TIMES LEFT
70C1 3EA5      01670      LO      A,0A5H  ; LOAD SYNC BYTE VALUE
70C3 B9        01680      CP      C      ; COMPARE AGAINST C
70C4 CB        01690      RET      Z      ; RETURN IF A MATCH
70C5 1BDE      01700      JR      REOSYN  ; BACK TO KEEP LOOKING
              01710      ;
              01720      ; #####
              01730      ; ROUTINE TO RESET INSIG FLIP-FLOP TO BE ABLE TO READ
              01740      ; #####
              01750      ;
70C7 F5        01760      INSIG     PUSH   AF      ; SAVE VALUE IN AF
70C8 3A3040    01770      LO      A,[4030H] ; GET SCREEN STATUS BYTE
70C8 03FF      01780      OUT      [0FFH],A ; RESET FLIP-FLOP
70C0 F1        01790      POP      AF      ; RESTORE AF STATUS
70CE C9        01800      RET         ; BACK TO READING ROUTINE
              01810      ;
              01820      ; #####
              01830      ; THIS ROUTINE WILL LOAD A BASIC PROGRAM WITH NO FILENAME
              01840      ; #####
              01850      ;
70CF F3        01860      TEMPER     OI      ; GET RIO OF BOTHERS
7000 2AB140    01870      LO      HL,[40B1H] ; GET TOP OF BASIC MEMORY
7003 E05BA440  01880      LO      OE,[40A4H] ; GET BASIC PROGRAM PTR.
7007 05        01890      PUSH     OE      ; SAVE IT A MOMENT
7008 E052      01900      SBC      HL,OE   ; GET MEHORY AVAILABLE
700A E5        01910      PUSH     HL      ; GET READY FOR TRANSFER
700B C1        01920      POP      BC      ; COUNT OF MEMORY IS IN B
700C E1        01930      POP      HL      ; BEGINNING OF BASIC = HL
700D C5        01940      PUSH     BC      ; SAVE AVAILABLE MEMORY
700E C0A570    01950      CALL     REOSYN  ; CALL READ SYNC ROUTINE
70E1 C1        01960      POP      BC      ; RESTORE AVAILABLE MEM.
70E2 C5        01970      TREAD     PUSH   BC      ; SAVE BYTE COUNT
70E3 C07C70    01980      CALL     READ   ; READ ONE BYTE
70E6 71        01990      LO      [HL],C  ; GET VALUE TO VIDEO MEH
70E7 C1        02000      POP      BC      ; GET BYTE COUNT BACK
70E8 08        02010      DEC      BC      ; REDUCE COUNT BY ONE
70E8 78        02020      LO      A,B     ; GET HIGH BYTE OF MEHORY
70EA B1        02030      OR      C      ; AND CHECK AGAINST LOW
70EB CA9719    02040      JP      Z,1997H ; ON ERROR IF TOO MUCH
70EE 7E        02050      LO      A,[HL] ; GET VALUE IN HL
70EF A7        02060      AND     A      ; TEST IF A ZERO
70F0 C20271    02070      JP      NZ,JUMP3 ; PAST FLASH IF OKAY
70F3 3A3F3C    02080      LO      A,[3C3FH] ; GET PLACE ON SCREEN
70F6 EEOA      02090      XOR     0AH   ; AND TOGGLE STAR & SPACE
70FB 323F3C    02100      LO      [3C3FH],A ; AND PUT BACK ON SCREEN
70FB 2B        02110      DEC     HL      ; GO BACK SPACE FOR TEST
70FC 7E        02120      LO      A,[HL] ; GET VALUE THERE
70F0 A7        02130      AND     A      ; TEST IF A ZERO ALSO
70FE CA0571    02140      JP      Z,JUMP4 ; GO TO ENO ROUTINE IF 0
7101 23        02150      INC     HL      ; BACK TO PROPER BYTE
7102 23        02160      JUMP3    INC     HL      ; READY NEXT HEH LOC'N
7103 1B00      02170      JR      TREAD   ; AND THEN GO BACK
7105 23        02180      JUHP4    INC     HL      ; GET NEXT MEHORY LOC'N
7106 AF        02190      XOR     A      ; LET A BE EQUAL TO ZERO
7107 77        02200      LO      [HL],A  ; AND PUT IT IN PLACE
7108 E05BA440  02210      LO      OE,[40A4H] ; GET START OF PRGRH PTR
710C 3EFF      02220      LO      A,0FFH  ; GET RESETING CODE
710E 12        02230      LO      [OE],A   ; PUT AT PROGRAM START
710F C0FC1A    02240      CALL    1AFCH   ; RESET ALL LINE NUHBERS
7112 23        02250      INC     HL      ; HL MOVED PAST PROGRAH
7113 22F940    02260      LO      [40FBH],HL ; SIMPLE VARIABLE POINTER
7116 C0C901    02270      CALL    01C8H   ; CLEAR THE SCREEN NOW
7118 C0611B    02280      CALL    1B61H   ; CLEAR ALL THE POINTERS
711C C0FE01    02290      CALL    01FEH   ; TURN CASSETTE OFF
711F C3CC06    02300      JP      06CCH   ; GO TO BASIC "READY"
              02310      ;
              02320      ; #####
              02330      ; THIS ROUTINE IS A GERHINAL ROUTINE TO CSAVE A PROGRAM
              02340      ; WITHOUT A PROGRAM NAME. FORHAT: /PUT. THE 500-BAUD
              02350      ; BLOCK SAVE CAN BE USED AS AN EXAMPLE OF HOW TO EMPLOY
              02360      ; A PROGRAM NAME IN SAVING A PROGRAM.
              02370      ; #####
              02380      ;
7122 F3        02390      TEHPEX    OI      ; GET RIO OF BOTHER
7123 3E2A      02400      LO      A,2AH  ; GET READY AN ASTERISK
7125 323E3C    02410      LO      [3C3EH],A ; PLACE STAR ON SCREEN
7128 323F3C    02420      LO      [3C3FH],A ; PLACE STAR NEXT TO IT
712B 2AA440    02430      LO      HL,[40A4H] ; GTART OF BASIC PROGRAM
712E E05BF840  02440      LO      OE,[40FBH] ; BOTTOM OF VAR. POINTER
7132 C05F70    02450      CALL    SYNCHR   ; WRITE LEADER AND SYNC
7135 7C        02460      L0000P    LO      A,H     ; GET CURRENT HIGH MSB
7136 BA        02470      CP      0      ; SAME AS TARGET HSB?
7137 C23F71    02480      JP      NZ,JUMP01 ; CONTINUE IF NOT SAME
713A 70        02490      LO      A,L     ; SAME HI MSB - READY LOW
713B BB        02500      CP      E      ; SAME AS TARGET LSB?
713C CA5671    02510      JP      Z,G00OUT ; DONE WITH SAVE IF SO
713F 7E        02520      JUHP01    LO      A,[HL] ; ELSE GET VALUE IN MEH
7140 A7        02530      AND     A      ; TEST IF A ZERO
7141 C24E71    02540      JP      NZ,JUHP02 ; JUST GO ON IF NOT
7144 F5        02550      PUSH     AF      ; SAVE VALUE IN A
7145 3A3F3C    02560      LO      A,[3C3FH] ; GET TOGGLE VALUE TO A
714B EEOA      02570      XOR     0AH   ; TOGGLE STAR & SPACE
714A 323F3C    02580      LO      [3C3FH],A ; AND PUT IT ON SCREEN
7140 F1        02590      POP      AF      ; GET VALUE BACK TO A
714E 4F        02600      JUHP02    LO      C,A   ; PUT BYTE IN C REGISTER
714F C00070    02610      CALL    WRITE   ; WRITE BYTE TO TAPE

```

case is \$5.00. Photo 1 shows the unit mounted inside the smallest Radio Shack equipment box.

Eight-level (eight bit) paper tape is capable of storing parallel bytes of data by means of holes punched in the tape. A smaller, ninth hole – placed between the third and fourth holes – provides a timing signal for the reading program.

The ninth hole also can be used as a data-ready signal. By the time the light just triggers the circuitry as it passes along the edge of the smaller hole, the larger holes are letting in plenty of light for the data to be stable, ready to read.

The TPR-1 comes ready to hook to a computer bus. Its output is in parallel, and all signals are tri-state. Because it uses only 12 mA, it's possible to run the reader directly from the TRS power supply.

Figure 9-4 presents the diagram of the TPR-1. The low-power CMOS integrated circuits U1 and U2 evaluate the state of the data as seen by the light-sensitive transistors and provide a parallel output. Part of U2 is also used to drive the LED, which blinks on whenever data is stable at the output of the reader.

Figure 9-5 is the TRS-80 interface schematic. Z1 and Z2 decode the port address 3F in order to activate tri-state buffer Z3. This separate port decoding is necessary because the TPR-1 was not designed with the READY line separately activated from the data lines. Were that the case, READY might be tested at all times. That way, data would only be input whenever READY indicated stable data. In its present configuration, however, a separate buffer must be used for the TPR-1 data lines.

Z4 is a flip-flop which produces an interrupt signal and sends it to the TRS-80 INT line; INTAK (interrupt acknowledge) is used to clear the interface flip-flop when data has been read. This configuration is similar to that used for the interrupt based real time clock (see Chapter 8 ).

The circuit can be wire-wrapped on a small piece of perfboard and mounted inside a case with the TPR-1. A detachable 40-pin cable can also be used to save a few dollars.

Listing 9-2 presents the software to read one page (256 bytes) of data into the TRS-80 and store it in memory. Recall that the interrupt patch point at 4012 is initialized with C9, a RETurn instruction. In its place, then, a patch must be made to one of three interrupt service routines which will read each byte of data as it becomes stable at the output of the TPR-1. Since the reader will not likely be a device used very

## A Paper Tape Reader

```

7162 23      02620      INC      HL      ; GET NEXT MEMORY LOC'N
7153 C33571  02830      JP        L0000P ; LOOP AROUND FOREVER
7166 C0FE01  02840      CALL     01FEH   ; TURN OFF CASSETTE
7158 C9      02650      RET              ; BACK TO MAIN ROUTINE
;
02660 ;
02670 ; #####
02680 ; REMEMBER: ENTRY POINT FOR SAVE ROUTINE IS TEMPEX
02680 ; REMEMBER: ENTRY POINT FOR LOAD ROUTINE IS TEMPER
02700 ; #####
02710 ;
02720 ;
0000      02720      END

8ITEX 7036 00670 00360
G0OUT 7156 02640 02510
INSIG 70C7 01760 01280 01360
JUMPU1 713F 02520 02480
JUMPU2 714E 02600 02540
JUMPU3 7102 02160 02070
JUMPU4 7105 02180 02140
L0000P 7135 02460 02630
READ 707C 01160 01980
REDBIT 709C 01430 01180 01570 01630
REDEX 708A 01280 01170 01560 01620
REOSYN 70A5 01540 01590 01700 01950
REX2 7080 01290 01310
RLOOP 707E 01170 01210
SYLOOP 7069 01010 01070
SYNCR 705F 00970 02450
SYNCLP 7086 01620 01660
TEMPER 70CF 01860
TEMPEX 7122 02390
TIMEX 7014 00440 00330
TREAD 70E2 01870 02170
WLOOP 7008 00330 00370
WRITE 7000 00280 01030 01090 02610

```

NOTES -  
1 OUTPUT DATA IS TRUE  
I.E. A "ONE" (HOLE) ON THE  
TAPE GIVES A HIGH (5V)  
VOLTAGE LEVEL OUTPUT  
2 MAXIMUM LOADING ON ANY  
OUTPUT IS TWO STANDARD  
TTL LOADS  
3 Q1 THRU Q9 + MRD150  
OPTICALLY MATCHED, DO NOT  
SUBSTITUTE

U1, U2 + 502

FIG 9-4

often, this kind of interrupt-hogging is not a problem. If you have a Teletype with a built-in paper punch, though, you might want to configure this interrupt software so it can be patched in only when it's needed.

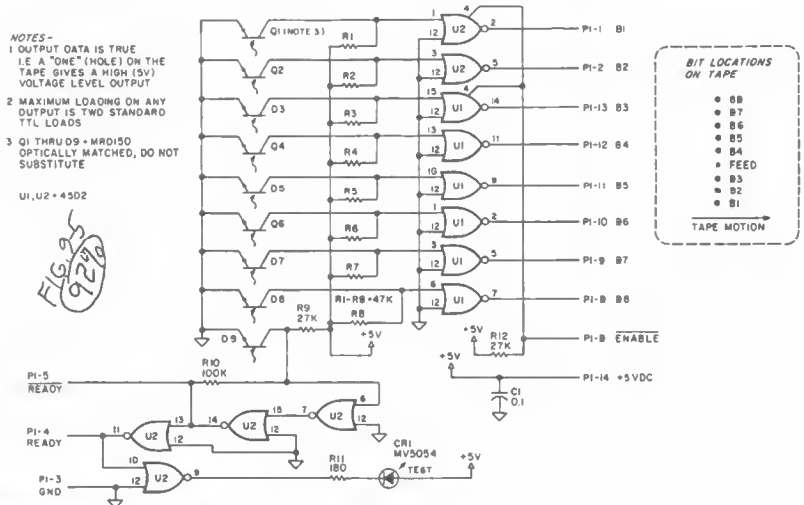
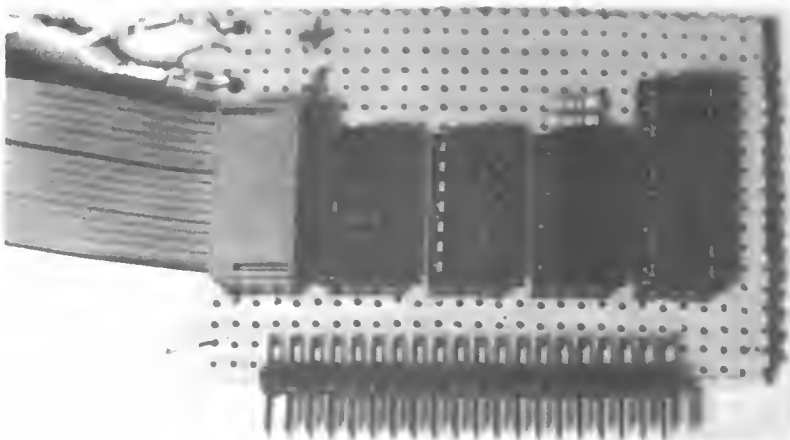


Figure 9-4. TPR-1 schematic diagram.



Photo 9-6. Paper tape reader.

TPR-1 tape reader has machined aluminum track and data indicator LED.



Only four integrated circuits form the complete tape reader interface. Power supplies both reader and interface.

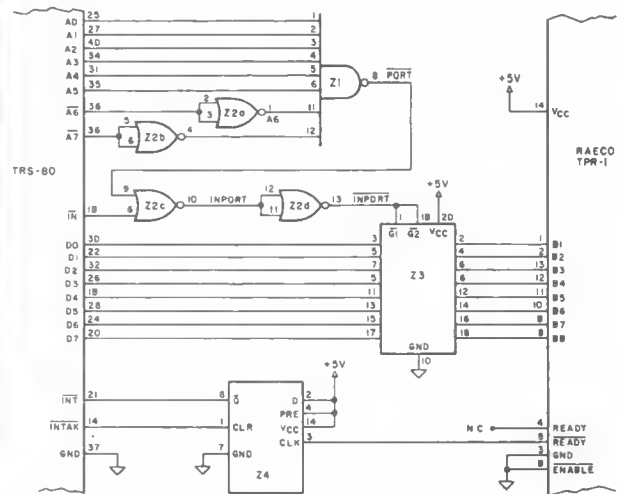


Figure 9-5. TPR-1 interfacing schematic.

The program is entered at line 1160. The screen is cleared, and the user is prompted to enter a base address in hex. This is the address starting at which the tape data is to be loaded into memory. The keyboard is scanned for characters 0 to 9 and A to F; these are displayed, and when ENTER is pressed, the characters are converted to a starting address.

The tape must be threaded before actual data reading is begun, because during threading it's possible to present false information to the TRS-80. The tape reading is begun at line 1740.

Listing 9-3. Paper tape reader program.

```

00100 ; #####
00110 ; TAPE READER AND 256-BYTE LOADER ROUTINE
00120 ; DENNIS BATHORY KITSZ, ROXBURY, VERMONT 05668
00130 ; #####
00140 ;
4013 00150 VECTOR EQU 4013H ; INTERRUPT VECTOR PATCH
06CC 00160 BASIC EQU 06CCH ; RETURN TO BASIC READY
3C00 00170 VIDEO EQU 3C00H ; BEGINNING OF SCREEN
00180 ;
00190 ; #####
00200 ; INTERRUPT VECTOR AT 4012H
00210 ; #####
00220 ;
4012 00230 ; ORG 4012H ; CURRENT VALUE IS C9
4012 C3 00240 0EFB 0C3H ; REPLACE WITH JUMP
7000 00250 ; ORG 07D00H ; ROUTINES BEGIN HERE
00260 ;
00270 ; #####
00280 ; CLEAR SCREEN SUBROUTINE
00280 ; #####
00300 ;
7000 21003C 00310 CLEAR LD HL,VIDEO ; GET START OF VIDEO
7003 11013C 00320 LD DE,VIDEO+1 ; GET DESTINATION POINT
7006 01FF03 00330 LD BC,03FFH ; GET MEMORY BLOCK SIZE
7008 3620 00340 LD (HL),20H ; WRITE SPACE INTO VIDEO
7008 EDB0 00350 LDIR ; BLOCK MOVE CLEAR SCREEN
700D CB 00360 RET ; BACK FROM SUBROUTINE
00370 ;
00380 ; #####
00380 ; SCAN FOR ENTER SUBROUTINE
00400 ; #####
00410 ;
700E 3A403B 00420 ENTER LD A,(3B40H) ; "ENTER" KEYBOARD ROW
7011 FE02 00430 CP 2 ; "ENTER" KEYBOARD COLUMN
7013 20F9 00440 JR NZ,ENTER ; LOOP UNTIL KEY PRESSED
7015 C9 00450 RET ; BACK FROM SUBROUTINE
00460 ;
00470 ; #####
00480 ; DISPLAY MESSAGE SUBROUTINE
00480 ; #####
00500 ;
7016 7E 00510 DISPLY LD A,(HL) ; GET BEGINNING OF TEXT
7017 A7 00520 AND A ; CHECK IF A NULL
7018 C8 00530 RET Z ; EXIT SUBROUTINE IF NULL
7019 12 00540 LD (DE),A ; DISPLAY CHARACTER IN A
701A 23 00550 INC HL ; GET NEXT MESSAGE LOC'N
701B 13 00560 INC DE ; GET NEXT SCREEN LOC'N
701C 18FB 00570 JR DISPLY ; LOOP FOR CHARACTER TEST
00580 ;
00590 ; #####
00600 ; CONVERT TO ASCII SUBROUTINE
00610 ; #####
00620 ;
701E F5 00630 CONVRT PUSH AF ; SAVE ACCUM. AND FLAGS
701F E8F0 00640 AND 0F0H ; MASK OUT LOW 4 BITS
7021 1F 00650 RRA ; MOVE NIBBLE TO RIGHT...
7022 1F 00660 RRA ; ...SOME MORE
7023 1F 00670 RRA ; ...SOME MORE
7024 1F 00680 RRA ; ...UNTIL OONE
7025 FE0A 00690 CP 0AH ; IS IT TEN OR GREATER?
7027 3004 00700 JR NC,HIBYTE ; MOVE ALONG IF > TEN
7028 CB30 00710 AOD A,30H ; ASCII = NUMBER PLUS 30H
7028 1B02 00720 JR NEXT ; GO ON TO LOW NIBBLE
702D C637 00730 HIBYTE AOD A,37H ; ASCII = NUMBER PLUS 37H
702F 77 00740 NEXT LD (HL),A ; DISPLAY FIRST ASCII CHAR
7030 23 00750 INC HL ; GET NEXT SCREEN LOC'N
7031 F1 00760 POP AF ; RESTORE ORIGINAL HEX
7032 E60F 00770 AND 0FH ; MASK OUT HIGH 4 BITS
7034 FE0A 00780 CP 0AH ; IS IT TEN OR GREATER?
7036 3004 00790 JR NC,HIBTE2 ; MOVE ALONG IF > TEN
7038 C630 00800 AOD A,30H ; ASCII = NUMBER PLUS 30H
703A 1B02 00810 JR NEXT2 ; GO TO DISPLAY & OUT
703C C637 00820 HIBTE2 AOD A,37H ; ASCII = NUMBER PLUS 37H
703E 77 00830 NEXT2 LD (HL),A ; DISPLAY NEXT ASCII VALUE
703F C8 00840 RET ; BACK FROM SUBROUTINE
00850 ;
00860 ; #####
00870 ; CLEAR TAPE READER INTERRUPT ACKNOWLEDGE
00880 ; #####
00890 ;
7040 F3 00900 SERVED DI ; INTERRUPT OFF IN SERVICE
7041 AF 00910 XOR A ; CLEAR ACCUM. & FLAGS
7042 C8 00920 RET ; BACK FROM SUBROUTINE
00930 ;
00940 ; #####
00950 ; PAGE ADDRESS INTERRUPT SERVICE
00960 ; #####
00970 ;
7043 F3 00980 SERVE1 DI ; INTERRUPT OFF IN SERVICE
7044 DB3F 00990 IN A,(3FH) ; GET VALUE FROM READER
7046 C01E7D 01000 CALL CONVRT ; CONVERT VALUE TO ASCII
7048 AF 01010 XOR A ; CLEAR ACCUM. & FLAGS
704A C9 01020 RET ; BACK FROM SUBROUTINE
01030 ;
01040 ; #####
01050 ; READ DATA / PLACE ON SCREEN INTERRUPT
01060 ; #####
01070 ;
704B F3 01080 SERVE2 DI ; INTERRUPT OFF IN SERVICE

```

Listing Continued . . .

With the software shown, the tape to be read must be in the following format:

1-byte code of information (tape number, address page, etc.), which is displayed but not to be stored in memory.

256 bytes of data.

1-byte simple checksum.

If the tape is not in this format, the program can be easily altered to accommodate any other 256-byte data block format.

Interrupts are then enabled (lines 1810-1820), and a series of short interrupt service routines are activated. The first routine merely waits for the interrupt line to clear, as it may have been set by stray light in the room when the tape is threaded (lines 1820-1870). 256 bytes are then loaded and displayed (lines 1910-2030). The checksum is calculated and displayed (lines 2050-2130), and the checksum is read from tape and displayed (lines 2150-2280). If there is a match, the memory pointer is advanced in order to read the next block of tape; otherwise, it is reset to the beginning of the block, allowing the tape to be read again. (lines 2210-2420). Finally, the option of loading additional blocks or returning to BASIC is presented (lines 2440-2540).

Using the TPR-1, the interface, and this simple software, the wealth of 8080 programs, as well as programs saved in an archival paper tape format, may be read into your TRS-80 and used.

## An 8-Track Mass Storage System

Oh, no! Here comes another one! I'd like to join the mass storage fray with another device capable of loading and saving programs at high speed. It's not as slow as a cassette, not as fast as a Stringy-Floppy, but it has one interesting capability: sequential-random access. That's a mythical term for sequential access of more than one track at a time.

Here's how it works: 8-track cartridges play one-quarter their total length on each pass. Then the head switches from the first stereo pair to the second, the second to the third, the third to the fourth, and from the fourth back up to the first. A so-called '40-minute' cartridge is actually 10 minutes long, four passes. The shortest commercially available cartridges are 20 minutes long, five minutes per pass.

## An 8-Track Mass Storage System

### Continued Listing

```

704C 0B3F 01090 IN A,(3FH) ;GET VALUE FROM READER
704E 77 01100 LO (HL),A ;PUT IT INTO MEMORY
704F B1 01110 ADD A,C ;GET VALUE FROM CHECKSUM
7050 4F 01120 LO C,A ;RESTORE UPDATED CHECKSUM
7051 23 01130 INC HL ;GET NEXT MEMORY LOC'N
7052 AF 01140 XOR A ;CLEAR ACCUM. & FLAGS
7053 C9 01150 RET ;BACK FROM SUBROUTINE
01160 ;
01170 ; #####
01180 ; CHECKSUM INTERRUPT ROUTINE
01190 ; #####
01200 ;
7054 F3 01210 SERVE3 OI ;INTERRUPT OFF IN SERVICE
7055 0B3F 01220 IN A,(3FH) ;GET VALUE FROM READER
7057 47 01230 LO B,A ;SAVE IT IN B REGISTER
7058 AF 01240 XOR A ;CLEAR ACCUM. & FLAGS
7058 CB 01250 RET ;BACK FROM SUBROUTINE
01260 ;
01270 ; #####
01280 ; MESSAGES FOLLOW
01290 ; #####
01300 ;
705A 54 01310 MSGN01 0EFH 'THREAD TAPE AND PRESS CLEAR.'
7076 00 01320 0EFB 00
7077 4C 01330 MSGN02 0EFH 'LOADING PAGE ADDRESS: '
708E 00 01340 0EFB 00
708F 42 01350 MSGN03 0EFH 'BYTES LOADING AS FOLLOWS:'
70A8 00 01360 0EFB 00
70A8 43 01370 MSGN04 0EFH 'CALCULATED CHECKSUM IS: '
70C2 00 01380 0EFB 00
70C3 43 01380 MSGN05 0EFH 'CHECKSUM AS READ IS: '
70D8 00 01400 0EFB 00
70DA 43 01410 MSGN06 0EFH 'CHECKSUM ERROR IN THIS BLOCK.'
70F7 00 01420 0EFB 00
70F8 42 01430 MSGN07 0EFH 'BLOCK LOADED CORRECTLY.'
70F0 00 01440 0EFB 00
7E10 41 01450 MSGN08 0EFH 'ANOTHER BLOCK? REPLY 1 FOR YES, 2 FOR NO'
7E39 00 01460 0EFB 00
7E3A 50 01470 MSGN09 0EFH 'PRESS CLEAR TO RETURN TO BASIC.'
7E58 00 01480 0EFB 00
01490 ;
01500 ; #####
01510 ; REMEMBER THIS IS ENTRY POINT AND NOT!
01520 ; BEGINNING OF PROGRAM.....
01530 ; CLEAR SCREEN, DISPLAY "THREAD" MESSAGE
01540 ; #####
01550 ;
7E5A C0007D 01560 START CALL CLEAR ;OUT TO CLEAR SUBROUTINE
7E5D 215A7D 01670 LD HL,MSGND1 ;GET MESSAGE #1 LOCATION
7E60 11003C 01580 LD DE,VIDED ;GET DISPLAY LOCATION
7E63 C0167D 01590 CALL DISPLY ;OUT TO DISPLAY SUBROUT.
7E66 C00E7D 01600 CALL ENTER ;WAIT FOR ENTER SUBROUT.
01610 ;
01620 ; #####
01630 ; DISPLAY "ADDRESS" MESSAGE & FIND IT
01640 ; #####
01650 ;
7E69 21777D 01660 LD HL,MSGND2 ;GET MESSAGE #2 LOCATION
7E6C 11403C 01670 LD DE,VIDED+40H ;GET DISPLAY LOCATION
7E6F C0167D 01680 CALL DISPLY ;OUT TO DISPLAY SUBROUT.
7E72 21407D 01690 LD HL,SERVE0 ;GET INT #1 SERVICE ROUT.
7E75 22134D 0170D LD (VECTOR),HL ;INSTALL AT INT. VECTOR
7E78 37 01710 SCF ;CARRY FLAG IS IMPORTANT
7E79 E056 0172D IM 1 ;SET INTERRUPT MODE
7E78 F8 01730 EI ;INTERRUPTS ON & WAITING
7E7C 3BFE 0174D JR C,$ ;SUBROUTINE CLEARS CARRYI
7E7E 21497D 0175D LD HL,SERVE1 ;GET INT #2 SERVICE ROUT.
7E81 22134D 0176D LD (VECTOR),HL ;INSTALL AT INT. VECTOR
7E84 21573C 0177D LD HL,VIDED+57H ;GET DISPLAY LOCATION
7E87 37 0178D SCF ;CARRY DETERMINES LOOP
7E88 F8 0178D EI ;INTERRUPTS ON & WAITING
7E89 3BFE 0180D JR C,$ ;SUBROUTINE CLEARS CARRYI
0181D ;
0182D ; #####
0183D ; DISPLAY "BYTES" MESSAGE & LOAD 256
0184D ; #####
0185D ;
7E8B 21B77D 0186D LD HL,MSGND3 ;GET MESSAGE #3 LOCATION
7E8E 11803C 0187D LD DE,VIDED+80H ;GET DISPLAY LOCATION
7E91 C0167D 0188D CALL DISPLY ;OUT TO DISPLAY SUBROUT.
7E94 21487D 0189D LD HL,SERVE2 ;GET INT #3 SERVICE ROUT.
7E97 22134D 0190D LD (VECTOR),HL ;INSTALL AT INT. VECTOR
7E9A 21003D 0191D LD HL,VIDED+100H ;GET FIRST DISPLAY LOC'N
7E9D AF 0192D XOR A ;CLEAR ACCUM. & FLAGS
7E9E 4F 0193D LD C,A ;CLEAR CHECKSUM REGISTER
7E9F D60D 0194D LD B,00H ;LOAD B REGISTER WITH 255
7EA1 37 0195D LDDP2 SCF ;INSTALL "JR C" LOOP
7EA2 F8 0198D EI ;INTERRUPTS ON & WAITING
7EA3 3BFE 0197D JR C,$ ;SUBROUTINE CLEARS CARRYI
7EA5 10FA 0198D DJNZ LOOP2 ;WRITE ONE PAGE TO MEMORY
0199D ;
0200D ; #####
0201D ; DISPLAY "CHECKSUM CALC" MESSAGE
0202D ; #####
0203D ;
7EA7 21547D 0204D LD HL,SERVE3 ;GET INT #4 SERVICE ROUT.
7EAA 22134D 0205D LD (VECTOR),HL ;INSTALL AT INT. VECTOR
7EAD 21A87D 0206D LD HL,MSGND4 ;GET MESSAGE #4 LOCATION
7EB0 11403E 0207D LD DE,VIDED+240H ;GET DISPLAY LOCATION
7EB3 C0167D 0208D CALL DISPLY ;OUT TO DISPLAY SUBROUT.
7EB6 79 0209D LD A,C ;GET CHECKSUM CALCULATION
7EB7 D5 0210D PUSH DE ;SAVE DISPLAY INFORMATION

```

Listing Continued . . .

For this system, the shortest 8-track cartridges are used in an 8-track deck with an *electrical* fast-forward mode. There are several loading options:

1. Load the next program on the tape from the current track. The machine fast-forwards to the next leader and loads the program.
2. Load the next program on the tape, with the track specified. The machine moves to the specified track, fast-forwards to the next leader, and loads the program.
3. Load the program specified from the track specified. The machine moves to the appropriate track and reads leaders (in fast-forward mode) until the program is found, and then loads it.
4. Load the program specified; the machine moves ahead and reads the directory immediately following the splice. The program is located and read.

In this way, where the locations of programs are known, they may be loaded immediately. Otherwise, the device is *somewhat* directory organized. I add this reservation because the tape is sequential and programs can't be killed easily unless the tape is re-organized. More on that later.

The advantage of this system is obvious: it provides somewhat faster access and loading than cassettes, and allows fairly fast search and storage. In the fast-forward mode, 20-minute 8-track tapes can be run through completely in less than two minutes. Worst-case program access is then two minutes – when you have just passed the program you want to load. Furthermore, eight individual programs can be stored parallel to each other on the cartridge's tracks.

As noted, to build this device, an electrical fast-forward is necessary. Check the manuals for a two-speed motor; the Craig model H240 playback-only deck is the kind I used. Some modifications are necessary to the tape recorder itself, and alignment is a bit more critical.

## Continued Listing

```

7E8B E1      02110      POP      HL      ;TRANSFER IT TO HL PAIR
7E8B C01E70  02120      CALL     CONVRT ;CNVRT. CHECKSUM TO ASCII
              02130 ; #####
              02140 ; #####
              02150 ; #####
              02160 ; #####
              02170 ; #####
7E8C 21C370  02180      LO      HL,MSGN05 ;GET MESSAGE #5 LOCATION
7E8F 11803E  02180      LO      DE,VIDEO+280H ;GET DISPLAY LOCATION
7EC2 C01670  02200      CALL     DISPLY ;OUT TO DISPLAY SUBROUT.
7EC5 F8      02210      EI      ;INTERRUPTS ON & WAITING
7EC6 37      02220      SCF      ;CARRY FLAG LOOP SET
7EC7 3BFE    02230      JR      C,$ ;SUBROUTINE CLEARS CARRYI
              02240 ; #####
              02250 ; #####
              02260 ; #####
              02270 ; #####
              02280 ; #####
7EC9 78      02280      LO      A,B ;GET READ CHECKSUM BACK
7ECA 05      02300      PUSH     DE ;STASH DE REGISTER PAIR
7ECB E1      02310      POP      HL ;TRANSFER TO HL PAIR
7ECC C01E70  02320      CALL     CONVRT ;CNVRT. CHECKSUM TO ASCII
7ECF 78      02330      LO      A,B ;GET VALUE AGAIN FROM B
7E00 88      02340      CP      C ;CHECK WITH CALC CHECKSUM
7E01 2808    02350      JR      Z,CKSHOK ;CHECKSUM OKAY IF A MATCH
              02360 ; #####
              02370 ; #####
              02380 ; #####
              02390 ; #####
              02400 ; #####
7E03 210A70  02410      LO      HL,MSGN06 ;GET MESSAGE #6 LOCATION
7E06 11C03E  02420      LO      DE,VIDEO+2C0H ;GET DISPLAY LOCATION
7E08 C01670  02430      CALL     DISPLY ;OUT TO DISPLAY SUBROUT.
7E0C 1809    02440      JR      LEAVE ;LOAD COMPLETE - GO OUT
              02450 ; #####
              02460 ; #####
              02470 ; #####
              02480 ; #####
              02490 ; #####
7E0E 21F870  02500      CKSHOK LO      HL,MSGN07 ;GET MESSAGE #7 LOCATION
7EE1 11C03E  02510      LO      DE,VIDEO+2C0H ;GET DISPLAY LOCATION
7EE4 C01670  02520      CALL     DISPLY ;OUT TO DISPLAY SUBROUT.
7EE7 21107E  02530      LEAVE LO      HL,MSGN08 ;GET MESSAGE #8 LOCATION
7EEA 11003F  02540      LO      DE,VIDEO+300H ;GET DISPLAY LOCATION
7EE0 C01670  02550      CALL     DISPLY ;OUT TO DISPLAY SUBROUT.
              02560 ; #####
              02570 ; #####
              02580 ; #####
              02590 ; #####
              02600 ; #####
7EF0 3A1038  02610      FINDYN LO      A,(3810H) ;GET 0-7 KEYBOARD ROW
7EF3 FE02    02620      CP      2 ;IS IT NUMBER ONE?
7EF5 CA5A7E  02630      JP      Z,START ;BACK TO START IF SO
7EF8 FE04    02640      CP      4 ;IS IT NUMBER TWO?
7EFA 2802    02650      JR      Z,00NE ;FINISHED ROUTINE IF SO
7EFC 18F2    02660      JR      FINDYN ;KEEP LOOKING IF NEITHER
7EFE 213A7E  02670      00NE LO      HL,MSGN09 ;GET MESSAGE #9 LOCATION
7F01 11403F  02680      LO      DE,VIDEO+340H ;GET DISPLAY LOCATION
7F04 C01670  02680      CALL     DISPLY ;OUT TO DISPLAY SUBROUT.
7F07 C00E70  02700      LOOKNG CALL     ENTER ;LOOK FOR ENTER SUBROUT.
7FOA C3CC06  02710      JP      BASIC ;BACK TO BASIC READY
              02720 ; #####
              02730 ; #####
              02740 ; #####
7E5A        02740      ENO      START ;SYSTEM ENTRY POINT
00000 TOTAL ERRORS

```

## Eviscerating an 8-Track Cartridge

Chances are that obtaining high quality 8-track cartridges in short lengths won't be easy. It's not hard to make your own in roughly eight minute lengths – two minutes per track, with less than 60 seconds total access time. This will be enough for programs nearly 30,000 bytes in length.

Purchase a few cheap 8-tracks to experiment with. All you will need is a piece of wood, a pair of scissors, and a package of silver foil sensing tape. This is available at Radio Shack (catalog number 44-1155, under \$2).

Most 8-tracks are fastened together with plastic tabs recessed in holes on top or bottom of the case. There are usually five: one on each back corner, one under the label in the center, one through the capstan, and one to secure the remaining corner. First, insert the cartridge into a player and run it ahead to its splice.

Make sure you are holding the cartridge with the label down. Slip a thin piece of hobby 'half-round' – a round stick split lengthwise down the center – into each hole and push the tab back, splitting the plastic case gently with your hand. The dull end of a small crocheting hook also works well. When all the tabs are released, split the cartridge apart to about one-quarter inch. Now *turn it over*. Gradually lift the cartridge apart, being careful to note the tape path and *exactly* how much slack is present in the loop.

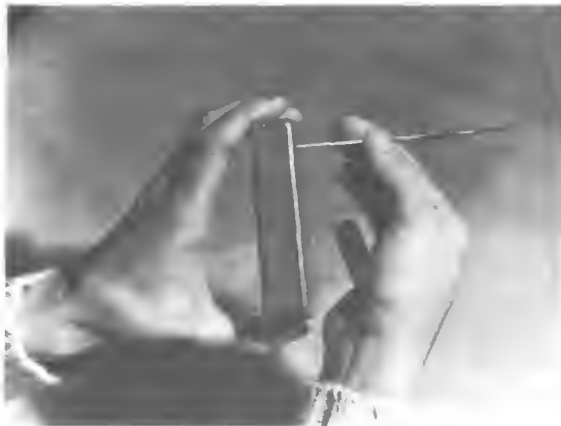
Snip the splice out first clipping out a length of tape that runs to within an inch of the center hub. Take the loose end of the tape that winds around the outside of the tape 'pancake', and begin to pull it out. Let the hub spin freely as you do this, so that the tension on the wound tape does not change. Measure the tape removed (to produce the 37.5 feet for an 8-minute cartridge) using this table:

Total Cartridge Time	Total Feet	Remove to Create 8-minute Cartridge
20 minutes	94	56
22 minutes	103	65
30 minutes	141	103
40 minutes	188	150
44 minutes	206	169
45 minutes	211	173
60 minutes	281	244

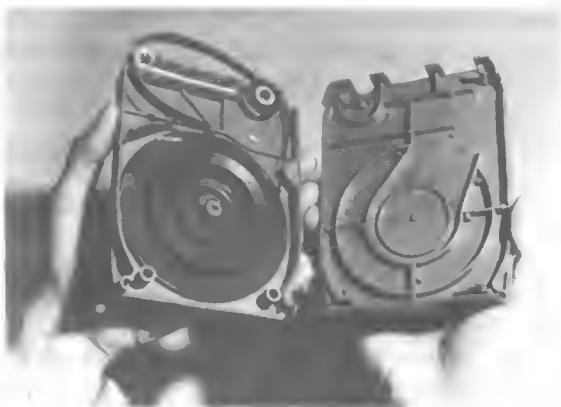
Note: Do not use cartridges over 60 minutes long.

Sound time consuming? It can be if you measure a foot at a time. Instead, anchor the cartridge on a table, and pull the end of the tape across the room (or down the hall). If you've purchased a 40-minute cartridge, that's only ten trips.

*Eviscerating an 8-Track Cartridge (5 photos).*



*Blunt tool is inserted in slots at bottom of the cartridge as the sides are separated and held steady.*



*Cartridge is flipped right side up and top of case is removed.*



*Tape is removed from its path and cut at splice.*

When you've removed all the tape you need to, leave the same amount of slack as before and splice the two ends together. The oxide (playing) side of the tape is spliced with metal foil sensing tape. Make sure you splice the same sides to each other! Now reroute the tape along the original path, holding it gently in place; use thin cloth gloves if your hands tend to perspire.



*Excess tape is pulled gently from outside of the hub, letting hub and platter rotate freely. Note that end near center of hub has been cut fairly short to eliminate the possibility of tangling with the tape being removed.*



*When enough tape has been removed, ends are spliced together with silver sensing foil. The tape is then re-routed and the case snapped back together.*

Without bending the tape or getting it caught, slip the cartridge top back in place. Note the front of the cartridge as you do this, making sure the tape does not slip on the outside of the 'window frames'. You now have an 8-minute cartridge, ready to use.

If you end up with some slack tape hanging out of the cartridge, put on some thin cloth gloves. Now pull the tape out of the case gently from the end opposite the rubber puck. The tape will pull into the other end faster than it is being pulled out. When most of the slack is taken up, give a gentle tug with a bit of 'snap'; the momentum will spool the tape into the cartridge housing.

If you are using the Craig H-240, the directions below will apply directly; since 8-track playback decks are similar, you can put some of them to use in remodeling your 8-track. It is also possible to purchase drives without electronics from surplus houses. These are sold for under \$10, and come complete with head assembly, motor, and track-change solenoid. Be sure that you get one with a two-speed motor if you want to implement the fast forward features of the system; an excellent single-speed drive is sold for only \$8 by BNF Enterprises.

First of all, you should know that the deck won't be usable as an audio deck when you're done; so don't split up your hi-fi system in hopes that this 8-track storage system will serve double duty.

1. Remove the case. Two screws hold the back panel, and screws through the four feet keep the frame attached to the housing. Set the back aside, and slide the electronics out through the front of the case.

2. Pull off the buttons, and set them aside. Remove the four screws which hold the frame to the face plate, and set the face plate aside.

3. Heat the soldering iron. You will be removing these wires:

- The two wires connected to the on-off switch; contact is made when a cartridge is inserted in the deck.

- The two wires from the foil sensor pickup. This is located about an inch from the on-off switch.

- All wires running to the three switches underneath the deck.

4. Remove the two front screws holding the three switches onto the front plate. Set this switch block aside; also remove all loose wires (those desoldered at both ends), and desolder the far ends of remaining wires which had run to the switches.

5. Unscrew the electronics control board (two screws), which is found to the front of the transformer. Desolder all wires leading to this board, and discard the board, scavenge it for parts, or keep it. It is a legitimate 8-track preamplifier, and can still be used if you need such an animal.

6. The following parts are still intact:

- The motor and the three wires leading from it. These wires are still attached to a terminal strip.

- The head assembly. This will be modified later. At present, it contains a shielded, three-wire cable leading from the playback head itself, and a five-wire assembly from the track-select switch.

- The capstan, drive belt, and track-change mechanics. These remain intact.

- The transformer. Three wires run from it; the center tap (black wire), won't be used, so cover it with tape or a wire nut.

- The terminal strip and two audio output jacks. These will be used.

By removing these parts, you have returned the drive to its 'naked' state. If you are using a surplus 8-track drive, this is the condition in which it will be shipped.

To use this in a digital system, several important conditions have to be met:

1. The recording and playback must be done in a digital format.

2. The track, splice, and tape-in-place status must be readable by the computer.

3. Speed must be controlled by the computer.

Figure 9-6. presents the complete circuitry to convert the Craig H-240 to a digital record/playback system. Incoming data is latched by Z12 on the occurrence of the command OUT (0AAH), A and is buffered by Z2a/b. It is fed to a symmetrical pair of output-coupled buffers (Z1 and Z6).

Since the output of these CMOS buffers is capable of rising very high (within a few millivolts of the 12-volt supply voltage), this provides a fast rising pulse to the recording head. The data is recorded in a bipolar manner: that is, when buffers Z1a/b rise, Z1c/d fall, and vice versa.

This information is recorded directly on the tape. During playback, the raw waveform is fed to Z4a, an LF353 FET operational amplifier (a plug-in replacement for the more commonly available LM747, which can be substituted with some signal degradation). This amplifier is set up in an inverted configuration with high gain; it produces a strong waveform which is then fed into Z4b, configured as a high-gain 'clipping' or 'squaring' amplifier (contributed by diodes D1 and D2).

This output is stabilized and buffered by comparator Z5a, and fed to Z5b (Z5 is a simple LM 339 comparator), arranged as a TTL-level driver. Z3a, a three-state inverter, is connected directly to a TRS-80 data line.

Figure 9-7 provides drive status information. Z10e is hooked to the former cartridge on-off switch, informing the computer of the presence of a cartridge in the drive. The track select switches feed their 12-volt signals to Z10a-b, which report the track pair in use. The foil sensor triggers a flip-flop made up of Z8a/b, which latches the fact that the foil has passed, until the computer resets it via the RESET SPLICE line.

The unit is turned on from the front, and power is always applied to the electronics. When the program reads or writes to tape, the motor is turned on via data bit 5; for fast read search and write, the motor is activated by data bit 6. The splice status is reset via data bit 3, and tracks are changed via data bit 4. Writing to the deck is enabled by data bits 1 and 2 (for the upper and lower of the track pairs, respectively), and the actual data writing is done through data bit 0.

Addressing of port AA is provided by Z14a-d and Z13, and this signal is combined with the computer's OUT and IN signals by Z8c/d. All the input data is latched by Z12.

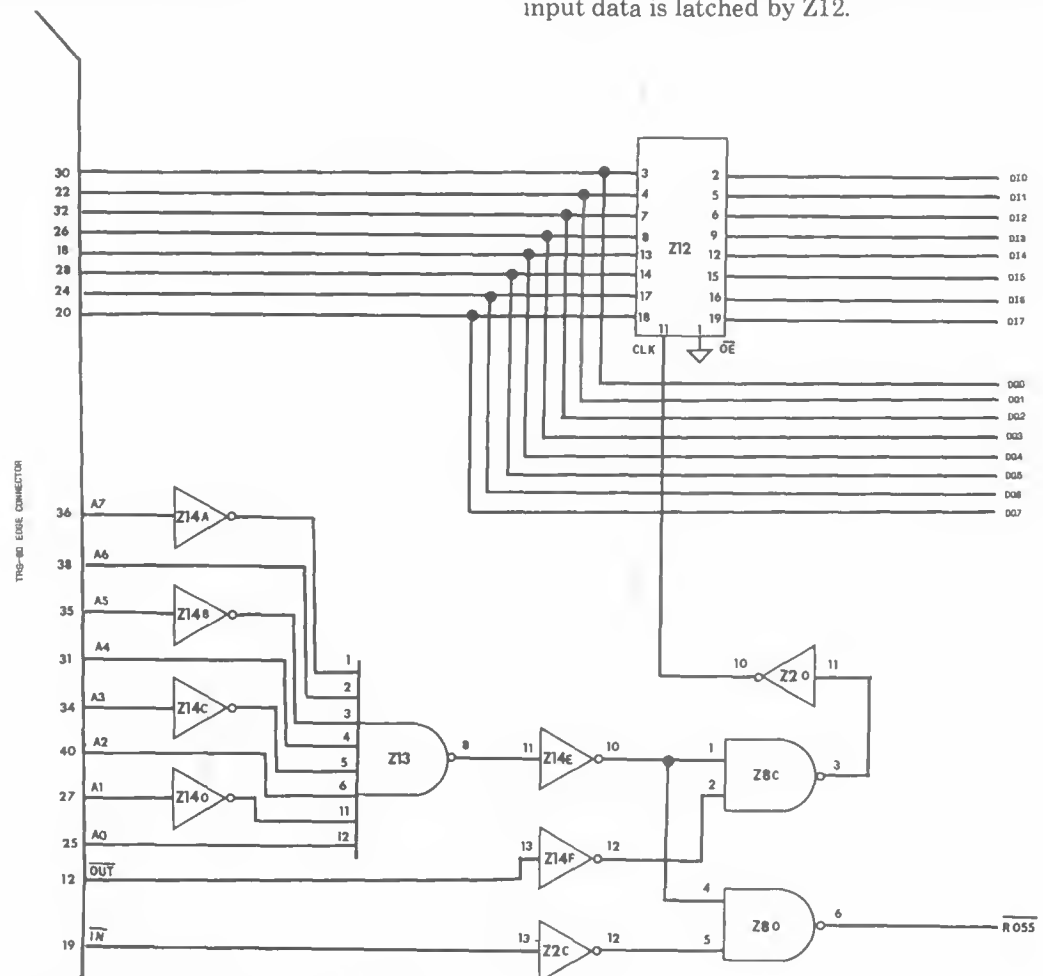


Figure 9-6. Schematic for 8-track storage system.

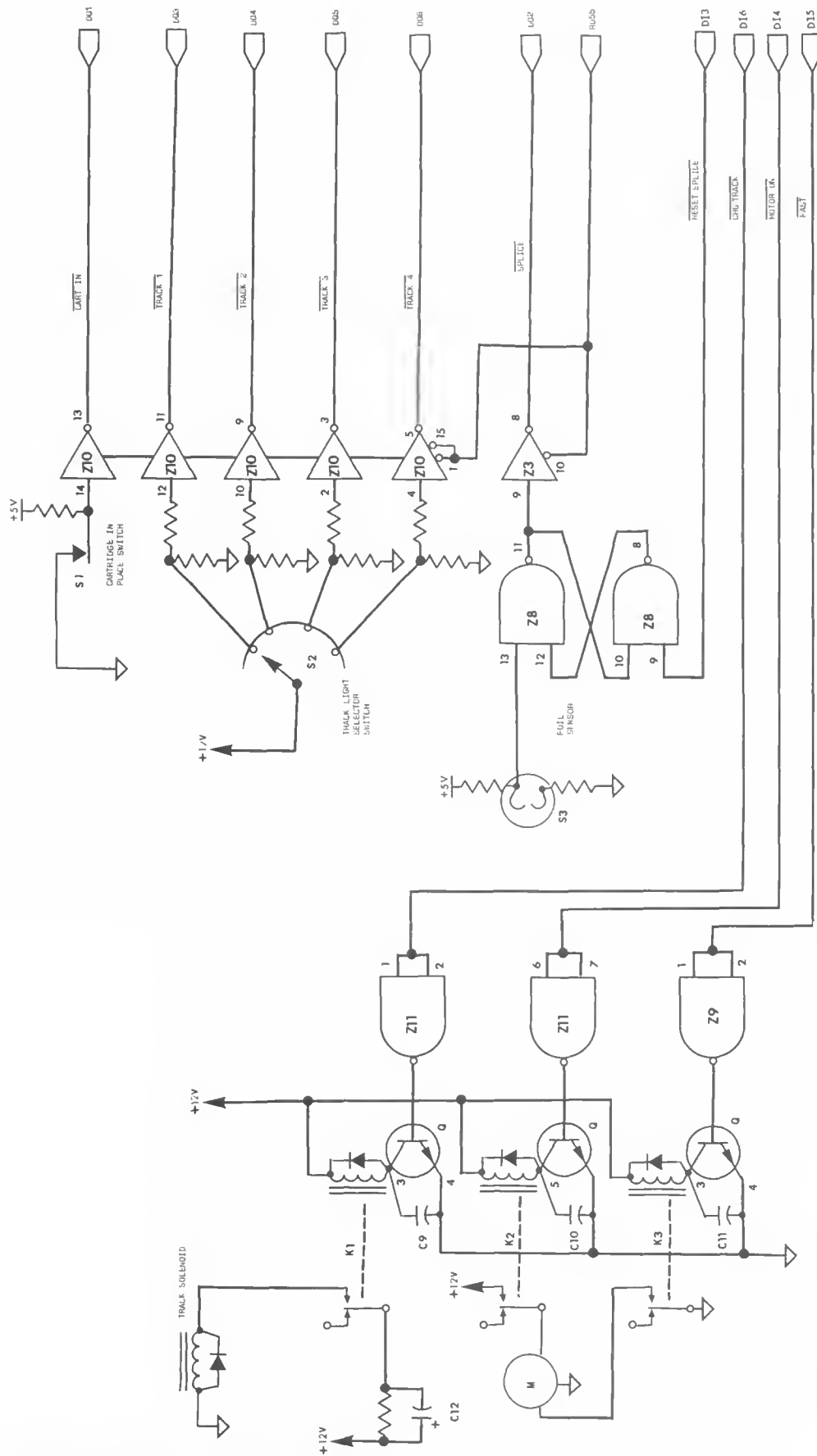
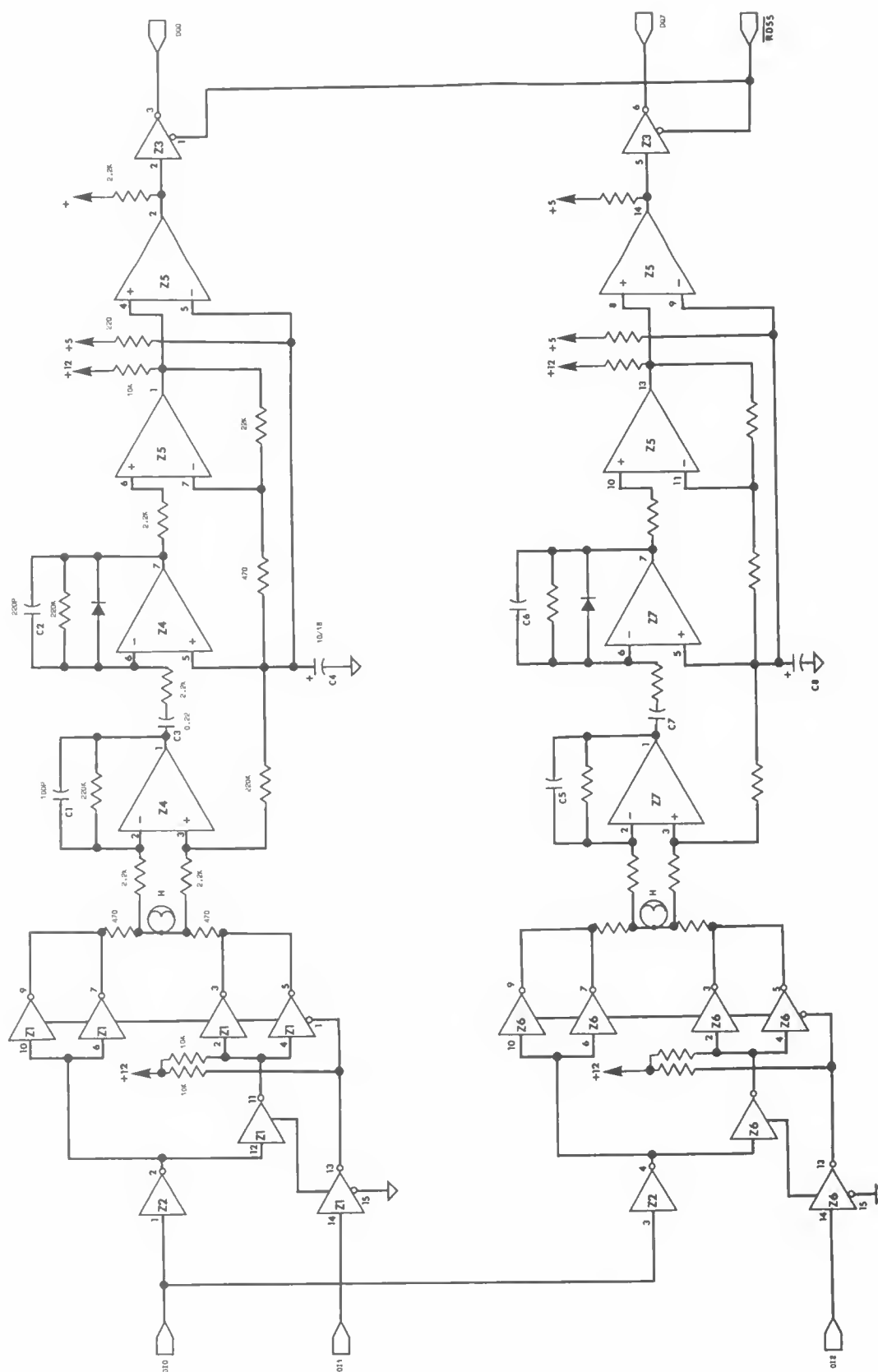


Figure 9-7. Schematic of tape player modification.



Listing 9-4. 8-track load and save cartridge system.

```

10 CLS : REM * THIS PROGRAM PRINTS CURRENT 8-TRACK DRIVE STATUS
20 OUTB5,16 : REM * START 8-TRACK DRIVE BY SETTING OUTPUT BIT 4
30 PRINT@0,"DATA 1:          DATA 2:"; : REM * L & R TRACKS
40 PRINT@128,"TRACK NO.:"; : REM * CURRENT HEAD POSITION PROMPT
50 A=INP(85) : REM * GET DATA FROM 8-TRACK DRIVE AT PORT 55 HEX
60 PRINT@66,(A AND 128)/128; : REM * TEST FOR HIGH DATA BIT 7
70 PRINT@66,(A AND 1); : REM * TEST FOR HIGH DATA BIT 0 & PRINT
80 B=(A AND 120) : REM * 120 = 78HEX = 01111000 TRACK CONITION
90 IFB=112THENPRINT@140,"1"; : REM * 01110000 = LO BIT 3 = TK1
100 IFB=104THENPRINT@140,"2"; : REM * 01101000 = LOW BIT 4 =TK2
110 IFB=88THENPRINT@140,"3"; : REM * 01011000 = LOW BIT 5 = TK3
120 IFB=56THENPRINT@140,"4"; : REM * 00111000 = LOW BIT 6 = TK4
130 IF(A AND 4)=4 THEN PRINT @ 256,"SPLICE DETECTED"; : GOT0190
140 IF(A AND 2)=2 THEN PRINT @ 320,"CARTRIDGE IS IN " ELSE
    PRINT @ 320,"INSERT CARTRIDGE ";
150 Q = RND(120) : REM * RANDOM TRACK SWITCH FOR TESTING ONLY
160 IF B=Q THEN OUT B5,B0 ELSE 30 : REM * SWITCH TRACKS HERE
170 FOR N = 1 TO 100 : NEXT : OUT B5, 16 : REM * RESUME NORMAL
180 GOT030 : REM * RANDOM TRACK SWITCH COMPLETE; BACK TO START
190 OUTB5,0 : A$=INKEY$ : IF A$="" THEN 190 : REM * OFF, THEN TEST
200 OUTB5,16 : GOT030 : REM * TURN BACK ON WHEN ANY KEY PRESSE0
210 OUT254,2 : REM * HIGH-SPEED SELECT FOR TESTING PURPOSES ONLY
220 CLS : REM * THIS ROUTINE CHECKS TOTAL CURRENT 8-TRACK STATUS
30 Q=148 : OUTB5,16 : REM * SET Q VALUE AND TURN ON TAPE DRIVE
40 A=INP(85) : REM * GET CURRENT STATUS OF TAPE DRIVE FROM PORT
50 B=(A AND 128)/128 : REM * GET VALUE AT DATA TRACK 1 (BIT 7)
60 C=(A AND 64)/64 : REM * GET VALUE AT TRACK POSN 4 (BIT 6)
70 D=(A AND 32)/32 : REM * GET VALUE AT TRACK POSN 3 (BIT 5)
80 E=(A AND 16)/16 : REM * GET VALUE AT TRACK POSN 2 (BIT 4)
90 F=(A AND 8)/8 : REM * GET VALUE AT TRACK POSN 1 (BIT 3)
100 G=(A AND 4)/4 : REM * GET VALUE OF SPLICE CONITION (BIT 2)
110 IF G=1 THEN OUT B5,0 : GOSUB 220 : REM * TURN OFF IF SPLICE
120 H=(A AND 2)/2 : REM * GET VALUE IF CARTRIDGE IS IN (BIT 1)
130 I=A AND 1 : REM * GET VALUE AT DATA TRACK POSN TWO (BIT 0)
140 PRINT@Q,"07 06 05 06 03 02 01 00"; : REM * PRINT DATA HEAD
150 PRINT@Q+64,B;C;D;E;F;G;H;I : REM * PRINT VALUES CALCULATED
160 PRINT@Q+192,"D T T T T S C 0 "; : REM * PRINT SOME
170 PRINT@Q+256,"A R R R R P A A"; : REM * PRETTY
180 PRINT@Q+320,"T K K K K L R T"; : REM * WORDS
190 PRINT@Q+384,"A # # # C T A"; : REM * FOLLOW
200 PRINT@Q+448,"1 4 3 2 1 E N 0"; : REM * STATUS.
210 GOT040 : REM * AND REPEAT THE PROCESS AS THE TAPE CONTINUES
220 OUT 255,255 : OUT 255, 0 : REM * A LITTLE SCREEN SHAKING
230 A$=INKEY$ : IF A$="" THEN 220 : REM * TEST, LOOP IF NO CHAR
240 OUTB5,24 : RETURN : REM * TAPE BACK ON, START PROCESS OVER

00100 : #####
00110 : SHORT 8-TRACK OPERATING SYSTEM FOR CONVERTED 8-TRACK
00120 : TAPE DECK. FORMAT:
00130 : ON, FAST FORWARD, FINO SPLICE, WRITE ENABLE, WRITE
00140 : 256 BYTES OF DIRECTORY ENTRY. LOOP UNTIL 10 ENTRIES ARE
00150 : WRITTEN OR NUMBER TRANSFERRED FROM BASIC IS ACHIEVED.
00160 : FILL OUT TRACK WITH FF'S, VERIFY WRITE (IF VERIFY FLAG
00170 : IS ON), FINO SPLICE, CHANGE TRACKS, REPEAT PRYESS FOR
00180 : ALL FOUR TRACKS.
00190 : DIRECTORY READ AT HIGH SPEED, ENTRIES ENTERED AT LOW
00200 : SPEED IN 256-BYTE PACKETS. EACH PACKET HAS A NUMBER
00210 : WHICH IS STORED IN THE DIRECTORY. NO DEFAULT ON TRACK
00220 : OR DIRECTORY ENTRY IS ALLOWED. DIRECTORY ENTRY #0 IS
00230 : RESERVED FOR FREE PACKETS ON EACH TRACK AND SECTOR.
00240 : *** NOTE: THIS LISTING IS NOT A COMPLETE OPERATING
00250 : SYSTEM. IT CONTAINS A SAMPLE DIRECTORY FORMATTING AREA
00260 : AND SAMPLE HIGH-SPEED TAPE READ/WRITE ROUTINES. A FULL
00270 : 8-TRACK OPERATING SYSTEM IS AVAILABLE; WRITE TO:
00280 : MSB ELECTRONICS, MR#1 DRAWER 766, BARRE, VERMONT 05641
00290 : OR DETAILS. THE INFORMATION PRESENTED IN THIS LISTING
00300 : IS FOR DEMONSTRATION, ALTHOUGH IT MAY BE ORGANIZED,
00310 : TOGETHER WITH OTHER STANDARD SAVE/LOAD PARAMETERS, INTO
00320 : A FULL OPERATING SYSTEM. ***
00330 : #####
00340 :
7000 00350 ORG 7000H
00360 :
7000 00370 R0BYTE EQU $ ; ***** REASSIGN TO READ BYTE
7000 00380 W0BYTE EQU $ ; ***** REASSIGN TO WRITE BYTE
7000 00390 LEADER EQU $ ; ***** REASSIGN TO LEADER WRITE
7000 00400 LEAD0R EQU $ ; ***** REASSIGN TO LEADER READ
0060 00410 DELAY EQU 0060H ; DELAY VALUE IN ROM
7000 00420 HOWMNY OEFB 100 ; 10 UNLESS BASIC XFER
7001 00430 VERFLG OEFB 00H ; TRANSFERRED FROM BASIC
0001 00440 VERFON EQU 01H ; STATUS REQUIRED BY TOS
0055 00450 PORT EQU 55H ; DEFINED AT B5 DECIMAL
0001 00460 WRITEN EQU 01H ; O10 = HIGH FOR WR ENABL
0002 00470 WRITEA EQU 02H ; O11 = HI/LO FOR WRITE A
0004 00480 WRITEB EQU 04H ; O12 = HI/LO FOR WRITE B
0008 00490 RESSPL EQU 08H ; O13 = HI TO RESET SPLICE
0010 00500 START EQU 10H ; O14 = HIGH FOR MOTOR ON
0020 00510 FSTFWO EQU 20H ; O15 = HIGH FOR FAST FWO
0040 00520 CHANGE EQU 40H ; O16 = HIGH TO CHNGE TRK
0001 00530 READA EQU 01H ; Q00 GOES HI/LO DATA A
0002 00540 CARTIN EQU 02H ; Q01 GOES HIGH AT CARTIN
0004 00550 SPLICE EQU 04H ; Q02 GOES HIGH AT SPLICE
0008 00560 TRACK0 EQU 08H ; Q03 GOES HIGH AT TRK 0
0010 00570 TRACK1 EQU 10H ; Q04 GOES HIGH A TRK 1
0020 00580 TRACK2 EQU 20H ; Q05 GOES HIGH AT TRK 2
0040 00590 TRACK3 EQU 40H ; Q06 GOES HIGH AT TRK 3
0080 00600 READB EQU 80H ; Q07 GOES HI/LO DATA B
7002 00 00610 STATUS OEF8 00H ; COMPOSITE OF ABOVE
00620 :
00630 : BASIC PATCH HERE — USE CUSTOM INTERPRETER
00640 :
00650 : #####
00660 : START 8-TRACK DECK MOTOR RUNNING (GIVE IT ENOUGH TIME)
00670 : #####
00680 :
7003 0E55 00690 AAAAAA LO C,PORT ; START VALUE
7005 3E10 00700 A,START ; BITS SET TO START

```

Listing Continued . . .

Figure 9- 8 presents the optional decoding of a ROM to contain the 8-track operating system. Notice that the decoding of the addresses is incomplete, so that data from 37C0 to 37FF must not be entered into the ROM to avoid bus conflict. The ROM should remain in its erased (all one's) condition in that memory area.

Listing 9-4 is an operating system for the 8-track storage system. It is made up of four major sections:

1. **Initialization.** Patching the operating system into the BASIC interpreter.
2. **Formatting.** The directory is set up past the splice on track zero in fast-forward mode. Each track is then written with program #1 headers. Because the tape is sequential, this is not a true disk-style directory. Instead, the directory stores the order and track number for each program, so that the correct track may be searched at high speed for its leader.
3. **Load Module.** This accepts the command, checks for correct syntax and program type, activates the tape deck and searches for the program.
4. **Save Module.** This module also accepts the command, checks syntax and program type, and writes the program to tape. The directory is updated.

The most interesting aspect of the software is the method used for recording the data. After a start level, a low is written to tape. Each subsequent bit changes the level either once (a zero) or twice (a one). Clock bits are not used in this scheme.

The operating system presented here is very basic, allowing only the elementary program save and load functions. However, data of all types may be stored in the system with additions to the software via additional commands.

Several problems may initially be encountered in using this system:

1. System does not respond. Make sure power is on; check for power to track lights. One should be lit. Find a cartridge which does not have its foil splice visible, insert the cartridge, switch to track 1 and PRINT INP (170). The value returned (in binary) should be x001000x. The x's are either one or zero, depending on the status of the data read outputs. Remove the cartridge. The value will change to x101000x. Switch to track 2, and replace the cartridge. Now it should read x000100x.

## An 8-Track Mass Storage System

### Continued Listing

```

7007 F620 00710 OR FSTFWO ; GET INTO HIGH GEAR
7009 320270 00720 LO (STATUS),A ; PUT IT INTO PLACE
700C 06FF 00730 LO B,OFFH ; GET LONG LOOPS
700E E079 00740 GOING OUT (C),A ; START 8 TRACK
7010 10FC 00750 QJNZ GOING ; MANY TIMES FOR SURE
00760 ;
00770 ;
00780 ; MOVE TO TRACK ZERO, CHECKING STATUS OF TRACK EACH TIME
00790 ;
00800 ;
7012 1600 00810 LO 0,0 ; TRACK VALUE TO CHECK
7014 E078 00820 LOOP1 IN A,(C) ; CHECK TRACK VALUE
7016 BA 00830 CP 0 ; AGAINST VALUE WANTED
7017 2800 00840 JR Z,JUMP1 ; GO IF TRACK = 0
7019 3E40 00850 LO A,CHANGE ; TRACK CHANGE VALUE
701B 2A0270 00860 LO HL,(STATUS) ; GET CURRENT STATUS
701E BA 00870 OR H ; CHECK PREVIOUS INFO
701F 320270 00880 LO (STATUS),A ; AND PUT INTO PLACE
7022 E079 00890 OUT (C),A ; CHANGE TRACK VALUE
7024 18EE 00900 JR LOOP1 ; GO BACK UNTIL TRK = 0
00910 ;
00920 ;
00930 ; FINO END-OF-TAPE SPLICE, AND DELAY TO GET PAST SPLICE
00940 ;
00950 ;
7026 E078 00960 JUMP1 IN A,(C) ; GET VALUE FROM PORT
7028 E604 00970 AND SPLICE ; CLEAR ALL OTHER VALUES
702A FE04 00980 CP SPLICE ; CHECK IF SPLICE DET'D
702C 20F8 00990 JR NZ,JUMP1 ; BACK UNTIL FOUND
702E E078 01000 JUMP1A IN A,(C) ; GET VALUE FROM PORT
7030 E604 01010 AND SPLICE ; CLEAR ALL OTHER VALUES
7032 FE04 01020 CP SPLICE ; CHECK IF STILL SPLICE
7034 28F8 01030 JR Z,JUMP1A ; BACK UNTIL SPLICE OVER
7036 010010 01040 LO BC,1000H ; GET DELAY VALUE
7039 C06000 01050 CALL DELAY ; DELAY TILL 6000 TAPE
01060 ;
01070 ;
01080 ; WRITE LEADER (NOTE: LEADER WRITE ROUTINE NOT INCLUDED)
01090 ;
01100 ;
703C C00070 01110 CALL LEADER ; WRITE LEADER
01120 ;
01130 ;
01140 ; FORMAT DIRECTORY (NOTE: THIS METHOD IS FOR EXAMPLE)
01150 ;
01160 ;
703F 210070 01170 LO HL,HOWMNY ; HOW MANY DIR ENTRIES?
7042 AF 01180 XOR A ; START WITH ENTRY #0
7043 06FF 01190 LO B,OFFH ; 256 BYTES TO WRITE
7045 C00070 01200 LOOP2 CALL WRBYTE ; WRITE THEM IN PLACE
7048 F5 01210 PUSH AF ; SAVE ACCUM TEMPORARILY
7049 E078 01220 IN A,(C) ; GET VALUE FROM DEVICE
704B E604 01230 AND SPLICE ; CLEAR ALL OTHER VALUES
704D FE04 01240 CP SPLICE ; SEE IF SPLICE HIT
704F 2802 01250 JR Z,JMPX ; GO IF SPLICE IS HIT
7051 10F2 01260 POP A ; ELSE ON TO NEXT BYTE
7053 F1 01270 JMPX LOOP2 ; RESTORE ENTRY VALUE
7054 3C 01280 INC A ; UP TO NEXT ENTRY
7055 BE 01290 CP (HL) ; LAST ENTRY COMPLETED?
7056 30E8 01300 JR NC,LOOP3 ; OUT IF NOT DONE
01310 ;
01320 ;
01330 ; FILL OUT TRACK WITH A SINGLE VALUE (AGAIN, OPTIONAL)
01340 ;
01350 ;
7058 3EFF 01360 LOOP4 LO A,OFFH ; WHEN DONE GET ALL FF'S
705A C00070 01370 CALL WRBYTE ; WRITE FF BYTE TO TAPE
705D E078 01380 IN A,(C) ; CHECK CONTENTS OF PORT
705F E604 01390 AND SPLICE ; MASK OUT OTHER BITS
7061 FE04 01400 CP SPLICE ; CHECK IF TO SPLICE
7063 20F3 01410 JR NZ,LOOP4 ; IF NOT KEEP WRITING
01420 ;
01430 ;
01440 ; GET NEXT TRACK VALUE AND GET TRACK FORMATTING READY
01450 ;
01460 ;
7065 14 01470 INC 0 ; GET NEXT TRACK VALUE
7066 7A 01480 LO A,0 ; CHECK CURRENT VALUE
7067 FE04 01490 CP 4 ; IS IT FOURTH TRACK?
7069 2819 01500 JR Z,JUMP2 ; IF NOT THEN BACK
01510 ;
01520 ;
01530 ; FORMAT NEXT TRACK AND CONTINUE UNTIL PROCESS COMPLETE
01540 ;
01550 ;
706B 3E40 01560 LO A,CHANGE ; GET CHANGE TRACK VALUE
706D 2A0270 01570 LO HL,(STATUS) ; GET CURRENT STATUS
7070 BA 01580 OR H ; STATUS PLUS CHANGE
7071 E079 01590 OUT (C),A ; AND CHANGE THE TRACK
7073 3E40 01600 LO A,CHANGE ; GET CHANGE TRACK AGAIN
7075 2F 01610 CPL ; SWITCH THE BITS
7076 2A0270 01620 LO HL,(STATUS) ; GET THE STATUS
7079 A4 01630 AND H ; STATUS PLUS CHANGE
707A 010002 01640 LO BC,200H ; GET DELAY VALUE
707D C06000 01650 CALL DELAY ; AND INVOKE THE DELAY
7080 E079 01660 OUT (C),A ; AND TURN IT OFF
7082 18A2 01670 JR JUMP1 ; AND GO BACK FOR MORE
01680 ;
01690 ;
01700 ; CHECK IF VERIFY FLAG IS ON (NOTE: ADD AS BASIC COMMAND)
01710 ;
01720 ;
7084 3A0170 01730 JUMP2 LO A,(VERFLG) ; STATUS OF VERIFY FLAG
7087 FE01 01740 CP VERFON ; SEE IF FLAG IS ON
7089 2036 01750 JR NZ,JUMP3 ; IF NOT THEN SKIP PAST
01760 ;
01770 ;
01780 ; MOVE TO THE FIRST TRACK, CHECK EACH AS IT PROGRESSES
01790 ;
01800 ;
708B 1600 01810 LO 0,0 ; ST COUNTER TO ZERO
708D E078 01820 JUMP4 IN A,(C) ; GET STATUS FROM DECK
708F BA 01830 CP 0 ; CHECK AGAINST TRK 0
7090 2818 01840 JR Z,JUMP5 ; IF AT ZERO, THEN GO
7092 3E40 01850 LO A,CHANGE ; ELSE BEGIN TO CHANGE
7094 2A0270 01860 LO HL,(STATUS) ; GET VALUE FROM STATUS

```

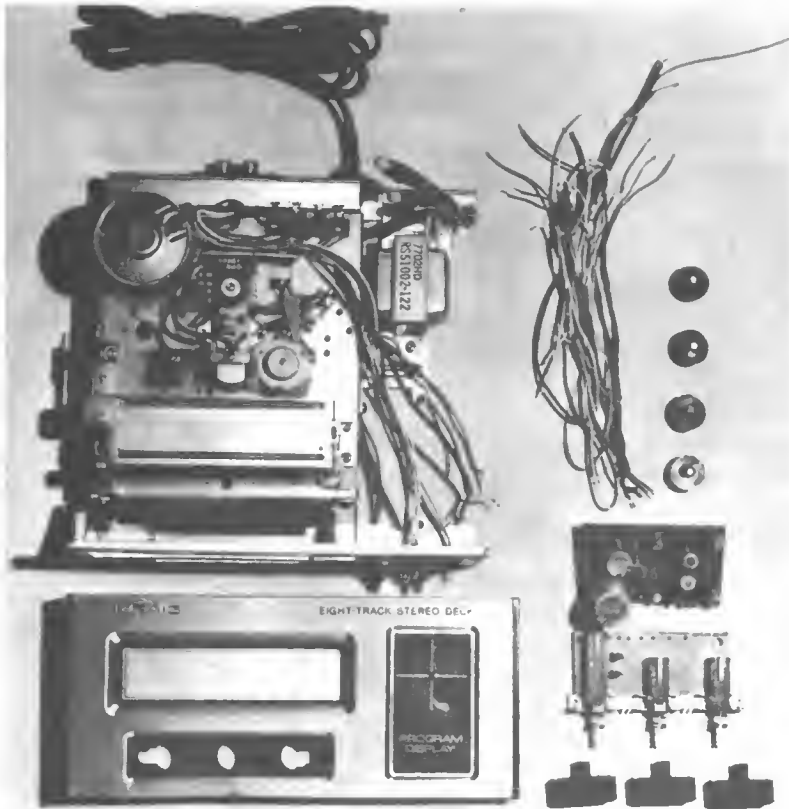
Listing Continued . . .

2. Tracks do not switch. Check wiring to the solenoid, and that the 75452 is wired correctly. Listen to hear if the solenoid attempts to react (a light click or start). Remove the 75452 from its socket and short the free lead of the solenoid to ground. It should switch. Replace the 75452 if necessary.

3. Programs do not load. If programs do not load at all, check the cartridge on an audio deck to see if something has been written. If not, go on to #4 below. If so, listen for occasional changes in pitch as the machine switches from fast forward to normal. Lengthen the speed change wait period in the program if you can hear the pitch slide as it restarts at a new level. If a loading message is displayed, but an error is detected, try to read from another track. Tracks 1a and 4b are at the edge of the tape, and lower-quality tapes may drop out occasionally in this area. The head may be badly misaligned and not make good contact with the tape. This can be heard as shifting or slewing in the sound. Adjust the Phillips alignment screw on the head to match a prerecorded commercial tape of good quality.

4. Programs do not save. Begin the program-saving process, and place the signal lead of a small amp against one lead of the recording head. If the signal is present, the program should be saving. If not, check the wiring of the buffer IC's, which may not be letting the signal through. Also check that the software is entered correctly, and that a signal is actually being sent to the device (correct connection of the write line, and proper wiring of the address decoding and data latch). If the motor turns on and switches tracks properly, the signal is probably being held up by incorrectly wired buffers to the recording head.

5. Motor speed does not change. Make sure that the third lead from the motor is being switched to ground, not positive voltage. This lead reacts best when switched below ground, and ground potential is its minimum position. If you have substituted another solid-state switch for the one shown, make sure it goes to full ground potential when switched in place.



Craig playback deck disassembled. Front panel, main frame and feet are maintained, but switch panel, playback electronics, wires, and buttons are discarded. Front buttons may be kept for appearance.

#### Continued Listing

```

7097 B4      01870      OR      H      ; AND CHECK STATUS VAL
7098 E079    01880      OUT     (C),A    ; AND SEND OUT CHANGE
709A 3E40    01890      LO      A,CHANGE  ; GET VALUE AGAIN
709C 2F      01900      CPL      REVERSE THE BITS
709D 2A0270  01910      LD      HL,(STATUS) ; GET VALUE FROM STATUS
70A0 A4      01920      AND      H      ; GET VALUE FROM STATUS
70A1 010002  01930      LO      BC,200H   ; GET DELAY VALUE
70A4 C06000  01940      CALL    DELAY    ; CALL DELAY IN ROM
70A7 E079    01950      OUT     (C),A    ; AND SEND OUT CHANGE
70A9 1BE2    01960      JR      JUMP4    ; AND GO ON THE REST
              01970      ; #####
              01980      ; #####
              01990      ; #####
              02000      ; #####
              02010      ; #####
70AB 1E00    02020      JUMP5    LO      E,0      ; PACKET NUMBER TO START
70AD 06FF    02030      LO      B,OFFH   ; BYTE TO VERIFY WITH
70AF C00070  02040      CALL    LEA0R0  ; GET SYNC AND READ LEA0R
70B2 C00070  02050      CALL    RDBYTE  ; BEGIN READING THE BYTES
70B5 BB      02060      CP      E      ; AND CHECK FORMAT INFO
70B6 202B    02070      JR      NZ,VERERR ; IF NOT CORRECT, ERROR
70B8 10F8    02080      DJNZ    LOOP6   ; AND GO BACK UNTIL DONE
70BA 1C      02090      INC      E      ; GO TO NEXT PACKET
70BB 3A0070  02100      LO      A,(HOWMNY) ; CHECK IF FORMAT DONE
70BE BB      02110      CP      E      ; BY GETTING PACKET #
70BF 20EC    02120      JR      NZ,LOOP7  ; BACK IF NOT DONE
              02130      ; #####
              02140      ; #####
              02150      ; #####
              02160      ; #####
              02170      ; #####
70C1 AF      02180      JUMP3    XOR      A      ; CLEAR ACCUM TO ZERO
70C2 320270  02190      LO      (STATUS),A ; PUT IN STATUS LOCATION
70C5 E079    02200      OUT     (C),A    ; AND SEND IT OUT PORT
              02210      ; #####
              02220      ; #####
              02230      ; #####
              02240      ; #####
              02250      ; #####
70C7 210070  02260      LO      HL,MESG01 ; GET THE OKAY MESSAGE
70CA CDA72B  02270      CALL    28A7H   ; LEVEL II DISPLAY ROUT.
70CC C37B10  02280      JP      107BH   ; BACK TO INTERPRETER
70D0 46      02290      MESG01 DEFM    'FORMATTING COMPLETE'
70E3 21EC70  02300      LO      HL,MESG02 ; VERIFY ERROR MESSAGE
70E6 CDA72B  02310      CALL    28A7H   ; DISPLAY SUBROUTINE
70E9 C3CC06  02320      JP      06CCH   ; BASIC COMMAND LEVEL
70EC 46      02330      MESG02 DEFM    'FORMAT ERROR'
              02340      ; #####
              02350      ; #####
              02360      ; #####
              02370      ; #####
              02380      ; #####

```

Listing Continued . . .

## Construction and Checkout

Since I recommend wire-wrapping the projects in this book, there are some different considerations when wrapping all the resistors and capacitors necessary for this project. Wire-wrapping is done best when there are sharp, square pins on which to wrap; since resistors and capacitors have round leads, it is best to insert these parts into sockets. Not only will the wrapping be more firmly attached, but the parts will not fall out when you are turning the board over during construction.

Before beginning construction, note that the transformer installed in the Craig recorder is marginal; ideally, it should be replaced with a 12 volt, 1.5 amp transformer of the type sold by Radio Shack. This will fit, though not comfortably, in the present transformer's location. The center tap is not used.

Because there are mechanical and inductive parts in this device, a great deal of electromagnetic noise can be produced. Be sure, then, to do the following:

1. Use bypass capacitors (.01 mF or so) between power and ground on every integrated circuit.
2. Use bypass capacitors (.01 mF) on the 75452 peripheral driver chips as noted, as well as diodes (hobbyist 1N914's are fine) across the three relay coils. Leave the diode already present on the track change solenoid. *Don't* skip these parts; they make the difference between a working system and a continuous return to MEMORY SIZE.
3. Use a central grounding point for all wires on the circuit board, and another central point for all wires on the drive. Don't weave wires from one IC to another.
4. The resistors at the inputs of the CMOS chips (the 80C96 and 80C98s) are vital for a reliable signal, and also prevent damage to the chips themselves.
5. The wires leading to the recording head should be twisted or shielded to prevent picking up extraneous signals. Remember also that the connections of the top and bottom pairs of pins on the head should be the same.

## Continued Listing

```

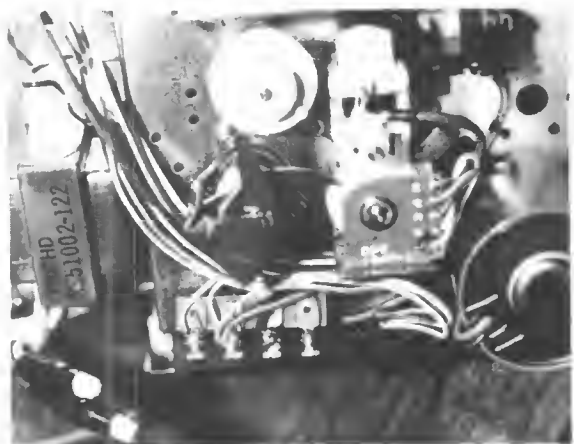
02390 ; RECORDING, WHICH WILL FUNCTION WITH A DIRECT DIGITAL
02400 ; RECORDING SITUATION SUCH AS THE B-TRACK DEVICE. FOR
02410 ; FURTHER INFORMATION ON ITS OPERATION, ASK FOR THE
02420 ; OHIO SCIENTIFIC TAPE WAFER INTERFACE, SOLD BY MSB
02430 ; ELECTRONICS (ADDRESS ABOVE), AND EXAMINE THE CODE.
02440 ; #####
02450 ;
02460 ; BOTTOM BASIC EQW 40FBH ; BOTTOM OF BASIC PROGRAM
02470 ; EQW 06CCH ; RETURN TO READY MOOE
02480 ;
02490 ; ENTER OI ; KILL THEM SUCKERS
02500 ; LO HL, (BOTTOM) ; GET FIRST PROGRAM LOC'N
02510 ; LO A, (4030H) ; VALUE IN THE MASK
02520 ; AND 0FCH ; SET BIT TO ZERO
02530 ; LO (4030H), A ; PUT BACK INTO LATCH
02540 ; LO E, A ; WILL BE USED A LOT
02550 ;
02560 ; NEXT LO A, (HL) ; SPECIAL TEST (EXAMPLE)
02570 ; LO 0, A ; THIS IS THE BIG BUMPER
02580 ; LO B, B ; NUMBER OF BITS TO WRITE
02590 ; LO A, 1 ; THIS IS THE START BIT
02600 ; OR E ; GET PROPER LATCH MASK
02610 ; OUT (OFFH), A ; WRITE OUT START LEVEL
02620 ; CALL DELAY1 ; WRITE A START DELAY
02630 ; XOR A ; SET UP A WITH 0 I/O BIT
02640 ; OR E ; GET PROPER LATCH MASK
02650 ; OUT (OFFH), A ; WRITE OUT STARTING EDGE
02660 ;
02670 ; #####
02680 ; ADDING 66 T-STATES HERE TO TOTAL 95 US
02690 ; #####
02700 ;
02710 ; PUSH IX ; 15 T-STATES
02720 ; POP IX ; 14 T-STATES
02730 ; PUSH IX ; 15 T-STATES
02740 ; POP IX ; 14 T-STATES
02750 ; NOP ; 4 T-STATES
02760 ; NOP ; 4 T-STATES
02770 ;
02780 ; #####
02790 ; THIS IS SET UP FOR 3300 BAUD
02800 ;
02810 ; TOTAL LOOP TIME PER BIT SHOULD BE 300 US
02820 ; TOTAL T-STATES FOR EACH BIT LOOP = 169 = 95 US
02830 ; TOTAL T-STATES FOR BIT DELAY LOOPS AT 0060 NOTED BELOW
02840 ; #####
02850 ;
02860 ; LOOP CALL DELAY2 ; WRITE A NORMAL DELAY
02870 ;
02880 ; XOR 0 ; TEST THE FIRST BIT
02890 ; AND 1 ; MASK OUT OTHER 0 BITS
02900 ; PUSH AF ; SAVE THE VALUE
02910 ; OR E ; VALUE OF LATCH MASK
02920 ; OUT (OFFH), A ; WRITE IT
02930 ;
02940 ; POP AF ; ORIGINAL VALUE BACK
02950 ; CPL 1 ; REVERSE THE BIT
02960 ; AND 1 ; MASK OUT ALL BUT ONE
02970 ; CALL DELAY2 ; WRITE A NORMAL DELAY
02980 ; PUSH AF ; SAVE THE PROPER BIT
02990 ; OR E ; AGAIN GET THE MASK
03000 ; LO (0000), HL ; NEEDED DELAY TIME
03010 ; NOP ; BIT MORE DELAY TIME
03020 ; OUT (OFFH), A ; WRITE IT
03030 ;
03040 ; POP AF ; GET ORIGINAL BITS BACK
03050 ; RRC 0 ; ORIENT TO NEXT BIT
03060 ; DJNZ LOOP ; WRITE OUT 8 BITS
03070 ;
03080 ; XOR A ; CLEAR ACC. TO ZERO
03090 ; OR E ; GET MASK FROM 4030
03100 ; OUT (OFFH), A ; SEND OUT ZERO BIT
03110 ; CALL DELAY2 ; SPACE OUT LAST BIT TOO
03120 ;
03130 ; INC HL ; GET NEXT MEMORY LOC'N
03140 ;
03150 ; #####
03160 ; PUT TESTING FOR MEMTOP HERE
03170 ; #####
03180 ;
03190 ; LO A, H ; CURRENT MEMORY STATUS
03200 ; CP 40H ; ENO? (EXAMPLE ONLY!!)
03210 ; JR NZ, NEXT ; GO BACK IF NOT DONE
03220 ; JP 06CC ; READY (USE RETURN!)
03230 ;
03240 ;
03250 ; LO A, H ; CURRENT MEMORY STATUS
03260 ; CP 40H ; ENO? (EXAMPLE ONLY!!)
03270 ; JR NZ, NEXT ; GO BACK IF NOT DONE
03280 ; JP 06CC ; READY (USE RETURN!)
03290 ;
03300 ; DELAY1 PUSH AF ; SAVE AF REGISTERS
03310 ; PUSH BC ; SAVE BC REGISTERS
03320 ; LO BC, 28H ; GET DELAY VALUE
03330 ;
03340 ; #####
03350 ; LINE ABOVE COMPLETES A TOTAL 1000 US DELAY LOOP
03360 ; #####
03370 ;
03380 ; CALL 0060H ; MAKE A DELAY
03390 ; POP BC ; RESTORE BC REGISTERS
03400 ; POP AF ; RESTORE AF REGISTERS
03410 ; RET ; BACK TO MAIN ROUTINE
03420 ;
03430 ; DELAY2 PUSH AF ; SAVE AF REGISTERS
03440 ; PUSH BC ; SAVE BC REGISTERS
03450 ; LO BC, 03H ; GET SHORTER DELAY VALUE
03460 ;
03470 ; #####
03480 ; LINE ABOVE COMPLETES AT TOTAL 205 US DELAY LOOP
03490 ; #####
03500 ;
03510 ; CALL 0060H ; MAKE A DELAY
03520 ; POP BC ; RESTORE BC REGISTERS
03530 ; POP AF ; RESTORE AF REGISTERS
03540 ; RET ; BACK TO MAIN ROUTINE
03550 ;
03560 ; #####
03570 ; ENO ENTER
03580 ;
070FB 00000 TOTAL ERRORS
21490 21490 TEXT AREA BYTES LEFT

```

Position sockets as close together as possible so the final board will fit into the tape drive's case. Fill the board with sockets and parts, and test its size before beginning the wire-wrapping.

Wire-wrap all connections completely before installing the 80C96 and 80C98 ICs. Because they are static-sensitive CMOS, they can be damaged by improper handling or application of power to partly-connected ICs.

When construction is complete, install the ICs, connect the unit to the TRS-80, and apply power.



Detail view of Craig mechanics. Track change mechanism is operated by a spinning cog on the drive capstan (lower right). Four track change lights are illuminated by switch contacts to the rear of the playback head.

```

AAAAA 7003 00690
BASIC 06CC 02470
BOTTOM 40F9 02460
CARTIN 0002 00540
CHANGE 0040 00520
DELAY 0060 00410
DELAY1 7143 03300
DELAY2 714E 03430
ENTER 70FB 02490
FSTFWO 0020 00510
GOING 700E 00740
HOWMNY 7000 00420
JMPX 7053 01270
JUMP1 7026 00960
JUMP1A 702E 01000
JUMP2 7084 01730
JUMP3 70C1 02180
JUMP4 7080 01820
JUMP5 70A8 02020
LEADER 7000 00390
LEADWD 7000 00400
LOOP 711F 02860
LOOP1 7014 00820
LOOP2 7045 01200
LOOP3 7043 01190
LOOP4 7058 01360
LOOP6 7082 02050
LOOP7 70A0 02030
MESG01 7000 02290
MESG02 70EC 02330
NEXT 7105 02560
PORT 0055 00450
ROBYTE 7000 00370
READA 0001 00530
READB 0080 00600
RESSPL 0008 00490
SPLICE 0004 00550
START 0010 00500
STATUS 7002 00610
TRACK0 0008 00560
TRACK1 0010 00570
TRACK2 0020 00580
TRACK3 0040 00590
VERERR 70E3 02300
VERFLG 7001 00430
VERFON 0001 00440
WRBYTE 7000 00380
WRITEA 0002 00470
WRITEB 0004 00480
WRITEN 0001 00460
02500
00850 01560 01600 01850 01890
01050 01650 01940
02620
02860 02970 03170
03570
00710
00750
02100
01250
00840 00990 01670
01030
01500
01750
01960
01840
01110
02040
03120
00900
01260
01300
01410
02080
02120
02260
02300
03270
00690
02050
00970 00980 01010 01020 01230 01240 01390
01400
00700
00720 00860 00880 01570 01620 01860 01910
02190
00560
00570
00580
00590
02070
01730
01740
01200 01370

```

---

# 10

---

## And Now It's Broken

It is not inevitable that your TRS-80 will fail during your lifetime, but there's always that chance. And if it happens, there's no reason to truck the computer down to the nearby Cost-a-Buck repair center. Do it yourself. This chapter will present the most likely failures or dilemmas you may encounter with your TRS-80, including:

**Setting** up a reliable, crash-free environment in a typically casual home.

**Curing** memory crashes in the CPU or the expansion interface, and replacing failed memory.

**Solving** the garbage-on-screen power-up failure.

**Discovering** the many sources of mysterious program crashes and keyboard lockup, and how to cure them.

**Aligning** your video display to cure images off-screen, tearing or jitter.

**How** program bugs can look like hardware failures, and vice versa.

**'Routine'** maintenance – the hidden cure for many failures.

**Handling** the computer and its peripherals.

**Overview** of difficulties in disk drivers, cables, cassette devices, printers, RS-232 boards, and other add-ons.

## A New Keyboard Cable

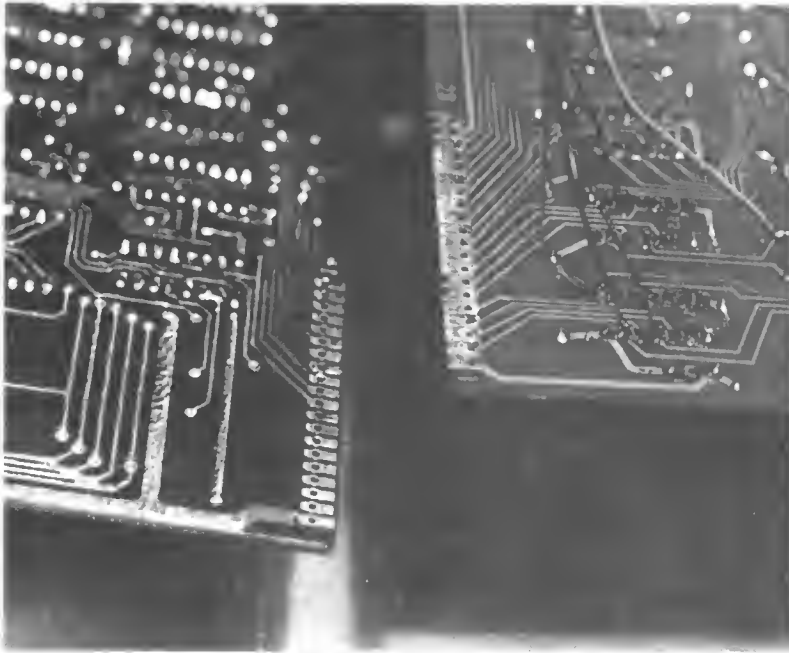
Virtually every modification to the TRS-80 contains a warning like this: 'Carefully open the case, and carefully take out the unit then carefully spread it out. Work carefully so as not to bust the keyboard cable'. So what's with the cable that makes it so fragile? The connections to the computer boards seem secure, but if you look carefully at the cable itself, you will see that it is made up of flat copper bands inside an insulating strip. The bands themselves are strong, and the connections to the computer are strong.

The problem occurs at the point where the copper bands are clamped to the connectors that attach to the circuit board. Hairline cracks develop in the copper bands, separating them almost invisibly from the connecting pins.

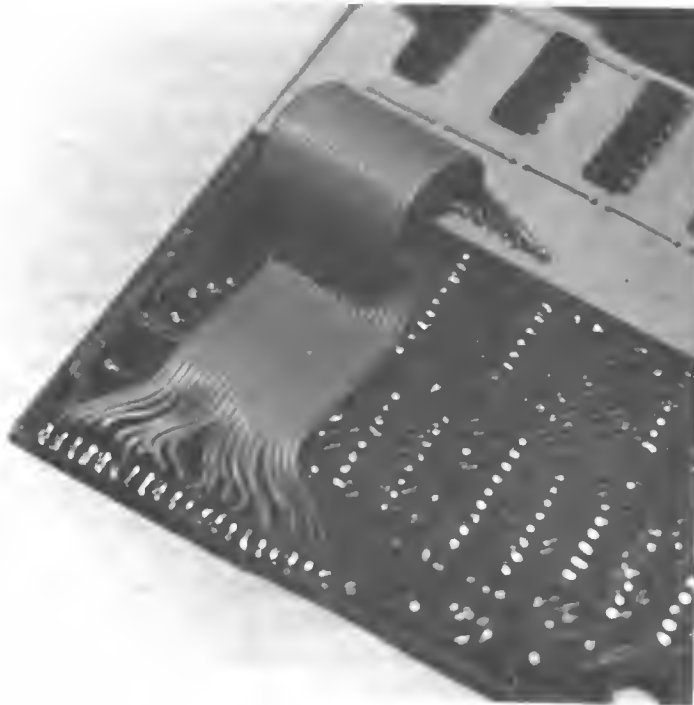
The only evidence of these cracks comes when odd combinations of letters appear sporadically while you are typing.

If you plan to open the machine more than a half dozen times, you should replace the connection, with a 20-wire multiconductor cable or very flexible single conductor phono cartridge wire.

First, cut the present cable off with shears, cutting at the point where the copper bands meet the connectors. Now, with generous amounts of solder-wick, remove the 40



*Replacing the keyboard cable: Solder-wick and flux cleaner are used to remove all traces of the old connection cable.*



*Replacing the keyboard cable: A new flexible cable is attached to both boards. A removable cable and connector can be used, but the permanent flexible cable serves well enough once most modifications have been made.*

connectors on the keyboard and CPU board. Make sure all the copper is out of the holes, and clean what is surely to be a mess with flux remover, then buff the solder so new wires will slip easily through the holes.

The new cable can be six inches long or more without affecting the operation of the unit. My own added keyboard, which is wired into place in the same manner as the original, has a 20 foot cable and suffers no ill effects or program crashes. Cut and strip no more than one quarter inch of insulation off each end, and tin all 40 ends.

Tinning is the process of running some hot solder onto the wire to prepare it for easy soldering to the board. The wire will remain the same diameter, the solder should not lump up, but will become smooth, hard and shiny with a solder coating. This will allow the wire to pass easily through the holes without random strands sticking out sideways and shorting against the neighboring connection. If the insulation creeps back from the heat, finish tinning all the ends and then clip them back to one quarter inch.

Pass the multiconductor through all the holes in advance, and secure it temporarily with masking tape or other tape not affected by warmth. Getting all the wires through can be tricky, but accept the absurdity of the process in advance; outsiders tend to burst into laughter at the sight of a grown adult trying to thread 20 needles simultaneously, so if you can't take a joke, do this with the door locked.

When all the wires are attached to either the keyboard or the CPU board (I recommend the keyboard first because it is lighter), check for shorts, then thread the cable through the other board. Solder, then look carefully for shorts before applying the power; both five volts and ground run through the cable, so be sure everything is well.

Apply power. All characters should work properly, and no odd combinations of letters should be produced. If the power LED does not come on, either a connection was left off or you have a short. Power off immediately, and check again. If letters are missing, a connection was probably not made. If odd combinations of letters appear, you probably have a broken wire. If incorrect letters appear, you have switched wires. Once it's all working, you can ignore those 'carefully . . .' warnings and concentrate on the modifications to be made.

## Home and Small Business Environments

Computers are designed in laboratories, tested in laboratories, examined by engineers and run by programmers. There could hardly be a more unlikely manner of producing an appliance-type product like the TRS-80. When you unpack the home computer and plug it in, you begin a torture test unimagined by the professionals in their sleek, air conditioned factories.

Below is a computer environment quick quiz; if you answer 'yes' to any question, your TRS could be in trouble. The more 'yesses', the more potentials for disaster:

- Your computer and peripherals are plugged into extension cords or cube-taps.
- A refrigerator, toaster, water pump, washer, dryer, or other large appliance is on the same fuse or breaker circuit as your computer.
- You have an electric mixer, blender, or food processor which is used when the computer is used.
- You have an electric drill, jigsaw, or table saw which is used when the computer is used.
- Your computer table desk lamp is a push-on and hold fluorescent type.
- You use your computer on or near a television set while the tv is turned on.
- Your computer is used in the kitchen or shop, you or a family member smokes, or you heat with wood or coal.
- You have fur-bearing animals in the house near your computer, especially cats.
- Your power is supplied by a rural cooperative, or is generated by a local power company especially with low-head hydropower systems.
- You live in an industrial area where heavy electrical equipment (winches, cranes) is used.
- Your electrical thermostat is located near the computer.
- Your home is especially dry, and you do not use a humidifier.
- You live in a coastal area, or by a salt-water lake.
- Your computer area is located near a railroad or by a highway traveled by heavy trucks.

- You move your computer while it is on.

Chances are you've got at least one check mark on the list above. Here are some solutions to these environmental problems:

1. Plug your computer directly into a wall outlet, or use a commercial 'power strip'.
2. Make sure that the computer is on a single-appliance circuit, even though it uses very little power.
3. Brush-motor electrical appliances like mixers and drills create an enormous amount of electrical noise. Reach a compromise with the culinary artist or the shop craftsman to use those tools at some other time, or at least far away from the computer.
4. Use incandescent lamps, which send out virtually no electrical hash, and get rid of the fluorescent ones. Don't use a television for a computer table, because it creates heavy electromagnetic fields. And move the computer or the thermostat; there's a tremendous electrical noise jolt transmitted when that thermostat turns on.
5. Keep smoke and grease of all kinds out of your computer's atmosphere, as well as animal (and people) hair. Tape and disk drives hate the stuff, and cables, connector, and keys build up greasy mudpiles because of them.
6. If your power is unsteady because of an inept or unconcerned power company, or because local industry unexpectedly drains a large amount of power, you will have to install some sort of power regulator. Types such as Solatron, Mayday, and Topaz provide different qualities of regulation, at corresponding costs (see Appendix).
7. Use a humidifier, or place pans of water on radiators or stoves if you house is especially dry in winter, because static electricity is quite powerful. If you live in a salt-air atmosphere, air-condition your computer area during warm, humid days. Salt air corrodes cables and connections.
8. If railroads or trucks are nearby, cushion the computer. Vibration can cause noise in cables and especially the expansion box, and make disk reading and writing very failure-prone. Likewise, moving the computer (or even just pulling or straining the expansion or peripheral cables) can create bursts of electrical noise through the computer. Don't do it. Interestingly, even

pushing away a hard chair that vibrates on a linoleum or hardwood floor can cause disk read/write errors. And don't forget the dummy plug for CTR-41 tape recorders when saving programs!

The home and small business environment is not often conducive to these suggestions, and some of them may not be necessary, depending on other factors. If you maintain your cables, edge card connectors, keyboards, etc., and keep your computer cushioned and seated on a conductive tabletop, you've gone a long way to increasing reliability.

Furthermore, merely thinking of your microcomputer in the same terms that used to be reserved for larger computers in the past (COMPUTER ROOM! NO SMOKING! . . . CAUTION! SENSITIVE ELECTRONIC EQUIPMENT! . . . NO FOOD OR DRINK IN THIS ROOM!), then you have the right idea. Electronically, your TRS-80 can take a great deal of abuse and still function. But this abuse cannot take place *while the computer is running*. And that is the clue: treat your operating computer as if you were paying \$50 an hour in time-sharing charges.

### When the Memory Crashes

In early TRS-80's, memory crashes were the most prominent sort of failure. The type of memories used were the culprit, partly because of expansion box design problems, but especially because an unusual condition called the 'soft error' had not then been diagnosed.

The 'soft error' was the tendency of a perfectly good program to crash with some error message when no such error was present. A simple CONTInue command would restore the system. These errors were caused by the internal structure of the memory chips themselves, which, because they are 'dynamic', use a peculiar and surprising principle for their operation.

Memories maintain information. That is their job. 'Static' memories retain information so long as the power is applied to the computer. 'Dynamic' memories, the type used in the TRS-80, retain their information for only a few thousandths of a second, requiring an electronic prodding, called *refresh*, to remember their data. They depend on their internal capacitance, acting much like a leaky tire.

In the early days, this odd way of maintaining memory resulted in occasional erratic behavior, sometimes because the chip itself was fluke, and sometimes because normal low-level radioactive

alpha-decay, *present right on the base of the chip*, could knock a memory bit from one state to another. This radioactivity is so delicate that a single sheet of paper can stop it. So this memory failure would occur only when the radioactive alpha particle actually struck a junction, and only when that junction was struck at precisely the right billionth of a second. Newer memories use a 'cool' base which does not emit alpha radioactivity, and so this rather bizarre problem has finally disappeared.

But memory crashes still occur, and they come in a few major forms:

**Temporary** crashes due to electrical noise in the vicinity of the computer.

**Temporary** crashes in the expansion interface due to a badly attached or otherwise noisy set of refresh lines.

**Temporary** crashes due to improperly seated or corroding memory chips.

**Permanent** crashes due to bad memory chips.

**Repairable** crashes due to a damage to one of the three lines responsible for memory refresh.

**Electrical** storms. I'll digress just a moment on this one. During a summer meeting of the Vermont Computer Guild, an electrical storm approached rapidly. We began to engage in a sly but nervous game of electronic chicken, leaving our micros not only attached to the power, but running! Naturally, that was too much of an invitation to Mom Nature, who zapped our very power line with a basketful of megavolts. My TRS-80 kept working; another hung, but reset. An Apple winked and sighed, and its memory cleared. A KIM turned tail completely, taking with it video routines and all. And the expensive DEC PDP/11 was broiled, losing more than a handful of expensive chips. Moral: You know it.

Also, there are crashes which appear to be memory crashes when they are in fact otherwise. Among these:

**A wayward** program containing errors in PEEK or POKE statements, or machine language subroutines.

**A damaged** CPU chip or blown buffer chip.

**A cracked** circuit board trace, solder ball, or solder splash.

### Cleaning the Edge Connectors

The expansion connector on the back of the CPU, and the various ports around the expansion interface, were all manufactured using a solder coating instead of gold plating. Because solder is lead, and lead tends to corrode badly, these connections will inevitably get electrically noisy.

Cleaning them is quite easy, and should be done, depending on your environment, from as often as weekly to a minimum of monthly.

Turn the equipment off and remove all cables. Check the edges of the cables for internal bend pins, hairs, or other damage or obstructions. Next, remove the top and bottom of the keyboard and expansion interface cases; if you don't wish to go inside your computer, this process can still be done, although it is

awkward.

Using a new dollar bill, a piece of the finest grade of emery paper, or a fine talc buffing wheel, bring the contacts to a bright shine. Brush off any remaining particles, and spray with contact cleaner (sometimes called tv tuner cleaner). Repeat this for all the edge connectors, and reinstall the cables.

If this process is repeated regularly, interconnect noise problems will be completely eliminated. However, if cables slide along the contacts when the computer is in use, noise may still be produced. To cure this, place no strain on the cables, and replace the keyboard expansion cable with a longer, more flexible unit. If your TRS-80 uses the buffered cable, instead of replacing it, merely add to it with one of the cable extenders sold by *Exatron* and others (see Appendix).

There are two software tools which can be used to eke out memory problems: the memory test printed in Chapter 3, and MEMORY SIZE? itself. Running the memory test will describe which memory locations and chips may be bad. When the MEMORY SIZE? question is answered with only a carriage return, a Level II subroutine begins testing each memory location until it finds a bad one; that final non-memory or bad memory location is found by typing:

PRINT PEEK [16561] + 256 \* PEEK [16562] + 2

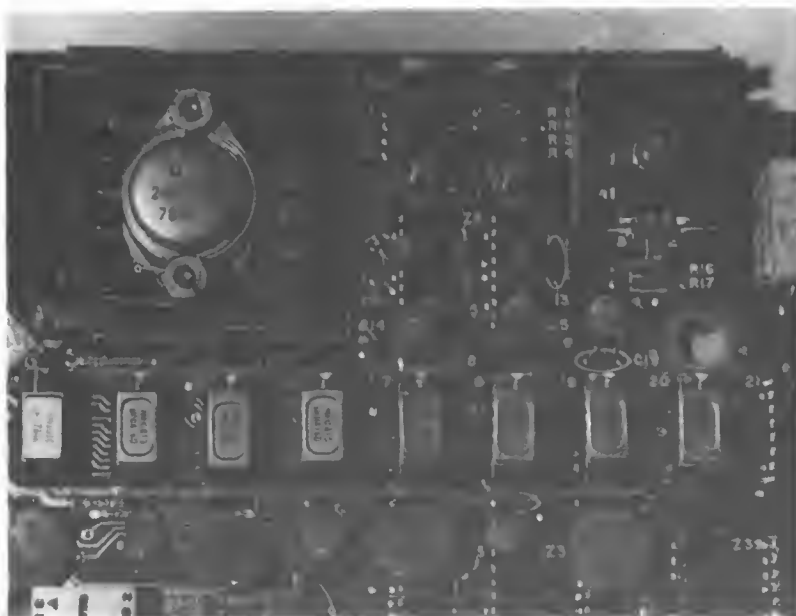


Photo 10-1. Power supply transistors.

In a 16K machine, the value returned should be 32768. If it is less, then the value returned is a bad memory location. Since the MEMORY SIZE? test is simplistic, it will not identify all possible memory errors, and if you suspect one, run the memory test.

If the bad memory location seems to move about, then perhaps it is not a true memory problem at all, but rather due to bad connections or electrical noise. Be sure noisy electrical equipment is not located near the computer. Clean cable and edge card connectors (see Box), and see if the problem recurs. If it does, open the case of your keyboard unit (and expansion box), and remove and reseat each memory chip. Look for corrosion, especially on newer chips without gold-plated leads.

If the problem is still not cured, the difficulty may be in the power supply. The location to suspect first is the large transistor (Q 4 – see

Photo 10-1), which is screwed down to the circuit board. Corrosion can build up between this transistor and the solder-plated circuit board. Loosen and remove the screws that attach the transistor, but do not attempt to remove the transistor itself. Slide fine emery paper – definitely *not* steel wool or sandpaper – face down between the transistor and the board, and clean out any corrosion.

If a lockwasher was not used between the transistor and the board, insert one, and reinstall and tighten the screws. This should stabilize the power supply inside the case.

## Using an Oscilloscope

The oscilloscope is a very sophisticated tool, but in this book it is used for an elementary purpose: merely to see if a signal is 'there' or not, and if it looks pretty good. Almost all TRS-80 circuitry failures can be traced this way, and it requires no previous experience using an oscilloscope, and no special training in reading waveform timings.

The height of an oscilloscope screen trace changes in proportion to the signal present at its input. To see this, plug in the scope, turn it on, and adjust the intensity until a flat line is visible in the center of the display. Attach its cables to the vertical input. Leaving the ground (black) wiring hanging, hold the signal (red) wire in one hand. The flat line displayed on the screen should go wild.

Adjust the vertical calibration (or voltage range) until you can see the trace. Adjust the sweep, sweep vernier and synchronization controls until the trace stabilizes, and looks like this:

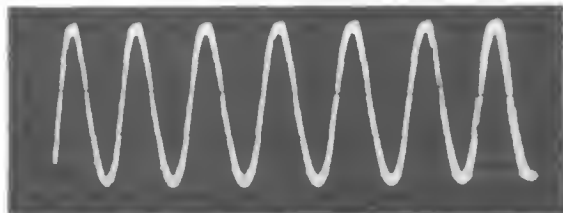


Photo 10-3. Photo of sine wave.

Adjust the vertical and horizontal centering until the trace is in the center of the screen. What you see is your own body acting as a kind



Photo 10-4. Photo of TRS-80 ground point for scope.

of transformer, soaking up all the electrical signals from the wiring and equipment all around you.

Now hold the black ground wire in your other hand. The wild trace should now flatten out considerably, as you become both signal and ground, essentially 'shorting out' the bulk of the signal received by the oscilloscope.

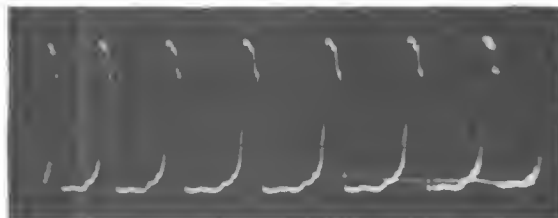
Next, power down your TRS-80 and open it. Using clips or by soldering a wire temporarily into place, attach the black ground wire to the point shown in the photo below:

Hold onto the red cable, and power-up the computer. Now find pin (?) of the Z-80 microprocessor (Z40 on the circuit board), and hold the red probe to it. Adjust the screen so you can see all of the trace, and it consists of a regular series of pulses.



Photo 10-5. 1.77 MHz clock pulses.

What you are looking at is the actual 1,774,000 pulse per second master clock of the TRS-80. All the signals you see throughout the computer should be of this quality – sharp-edged and clean. They will not have the same regularity, because different lines are turned on and off only as needed. But the signals should always be of the same crispness.



For practice, place the red probe at various points on the computer, being very careful not to short two points together with the tip of the probe. You will see the many kinds of signals present throughout the computer. If your computer has a problem, you will be looking for signal lines which have failed: they will be rounded, or angled instead of vertical, or will be much lower in level than the ones you see now. This is a clue to the area of the failure.

The next step is to switch the external black power supply box with a new one, because your present supply may have a damaged, highly resistive, or intermittent fuse wire. Later supplies can be opened to check this; there are screws under the feet. If the new supply works, have the old one repaired, or open it yourself and replace the fine fuse wire with a new one.

Memory problems caused by damaged refresh lines are subtle, but can be discovered. Load a BASIC program, LIST it, and re-LIST it. If gradually the program shows changes, suspect that the refresh lines may be weak. To diagnose this job fully, you will need an oscilloscope.

Photo 10-2 shows how cleanly a good signal is represented on the scope screen. Anything less than a sharp-edged, rectangular pulse should be suspect. Furthermore, a signal that is distinctly lower than others can be suspected.



Photo 10-2. Digital scope photo.

## Aligning the Video

Video jitter in most cases is due to incorrect settings of either the vertical or horizontal controls on the back of the monitor itself. To align these, enter the following short program:

```
10 CLS
20 FOR X = 15360 TO 16383
30 POKE X,191
40 NEXT
50 GOTO 50
```

Listing 10-1. Video centering routine.

This will paint the screen white. Any bulging on either side of the screen can be cured by adjusting the horizontal control on the back of the monitor. Turn it until the bulging is reduced as much as possible. Instability or rolling is most easily cured by adjusting the vertical control on the back of the monitor. 'Zero' it in by making the screen first roll forward, then back, then forward, and back again until you have what feels like the central position of the control.

Open the TRS-80, and set the program above in motion again. The horizontal and vertical positioning controls are located at the far right of the circuit board as shown in Photo 10-1. Adjust these until the image is properly centered, and drop a very small amount of glue or nail

polish on the control's plastic handle to hold it in that position against the control's body.

Any remaining instability in the screen display can usually be attributed to low voltage present on your house current line, or problems with the TRS-80 power supply (see above).

## Routine Maintenance

Microcomputers like the TRS-80 are generally thought of as maintenance free, except for peripherals with moving parts. But in truth, the keyboard unit and expansion box always need some maintenance, all of which is covered throughout this book. In summary:

1. Clean edge card connectors and check cables regularly.
2. Remove keytops (only on units with the older keyboards susceptible to keybounce) and blow out dust and clean contacts regularly.
3. Blow out dust from the unit itself, and give it a shake, to remove any solder balls and splashes that may come loose as the unit gets older.
4. Keep the unit covered when not in use.

## Care of Peripherals

**Cassette players.** Using ordinary rubbing alcohol and cotton swabs, clean the head, capstan and pinch roller to remove all traces of brown tape oxide. Demagnetize the head using a commercial demagnetization cassette, or a head degausser (available at Radio Shack); follow the directions carefully so you don't make the problem worse.

Always handle the player's buttons gently, as snapping them vigorously will not only shorten their life, but create enough vibration to throw the head out of alignment. If the head does get misaligned, or you suspect it is, turn to the details of cassette player maintenance in Chapter 6.

In summary: clean all parts that come in contact with tape, and treat the unit's controls gently.

**Printers.** Dot-matrix types like the Radio Shack line printers are workhorses, but there are a few cautions. First and foremost, never move or remove paper when the printer is in motion. The printed characters are formed by fine wires striking against the paper, and these wires can be bent if the paper is pulled against them.

Keep ribbons in good condition, because a ribbon with a slight tear can also catch on a head

and damage the dot-matrix wires. Open the printer and blow out dust and dirt regularly; if you are skilled in handling small machinery, lubricate the printer according to the Radio Shack maintenance manual. If you are a klutz, don't try; alignment of printers has to be accurate to 1/100th of an inch.

In summary: keep printers free of dust and ribbons fresh.

**Disk Drives.** These horrors are terribly sensitive. Like tape machines, the heads need cleaning, but use only head cleaning diskettes especially made for the purpose. If you have regular data read/write problems, there are several solutions: purchase a better disk operating system (DOS), and/or purchase and install an external data separator.

Keep out of the drive's insides as much as possible. Alignment is sometimes vital, but again, unless you are very good with precision machines, don't try it yourself. And even if you are a born watchmaker, be overly careful with double-sided or double-density drives.

In summary: beyond a regular cleaning, leave them alone. Oh yes, keep the drive cables well away from the monitor or any AC power cables.

**Exatron Stringy-Floppy.** Despite the manufacturer's warnings, this unit does not come well aligned from the factory, and many units were shipped without head alignment cement (normally Lok-Tite, a grey compound) to hold heads firmly in place. Remove the cover of the unit and check for this gop. If it is not there, turn to Chapter 9 and follow the alignment directions.

If the gop is not there, you'll need an oscilloscope. Turn the scope on, and adjust the vertical calibration for full scale (maximum). Hook the scope's ground (black) lead to a convenient ground point on the ESF or the TRS-80, and place the hot (red) probe on Z6 — pin 7. Start the ESF reading a prerecorded tape from the manufacturer. If the wave won't stay in synchronization on the screen, or if it is outside the vertical bounds of the screen, adjust the scope until it is visible and stable. With a small screwdriver, adjust the playback head's alignment until the amplitude (height) of the waveform is at its maximum. Try several prerecorded tapes to make sure of this position, and put some lok-tite on it. If you have your own tapes recorded on the ESF, you may have to move back and forth to and from the correct position, reading in the original position, and re-recording in the corrected one.

Clean the heads and capstan with a very sparing and gentle application of rubbing alcohol on the end of a lint-free swab. Don't use Q-tips or their equivalent, but rather wrap surgical gauze around a lollipop stick. Blow dust out of the unit regularly. Remember never to operate the ESF in any position but on its feet.

In summary: blow out dust, clean heads, and check for the presence of 'Lok-Tite' on the head screws.

**RS-232 Board.** In another piece of dream engineering, the connection of this board to the expansion box was done via a flukey, solder-coated connector board pressed against a plastic bridge with plain metal contacts. Clean those contacts regularly with a fine cloth, and brush the solder-coated board connectors vigorously with a buffing cloth or extra-fine emery paper. Spray both sparingly with tuner cleaner, and quickly reattach them, constantly rocking the board to assure a firm connection until you have tightened the screws down.

In summary: clean both sets of contacts occasionally, especially if you notice more than the usual transmission/reception errors.

## Diagnostic Programs and Loops

Loading large-scale programs for diagnosis is not only time-consuming, but as often as not impossible when your computer is not working well. If BASIC can be brought up at all, then simple machine language diagnostics can be POKEd into place from command level. This section will present several of those loops, their use in trapping some of the possible difficulties, and how they can be used to examine the operation of the system. All these routines can be POKEd anywhere in your machine's memory; these examples all start at 5000 (20480 decimal).

### 1. Checking the Write Circuits.

This routine merely writes to a location and jumps back to itself:

```
Coding:
AF      XOR    A
32 00 3C  LO    (3C00),A
3C      INC    A
18 FA    JR     $-4
```

Listing 10-2. Write circuit diagnostic/machine.

```
From BASIC:
X=20480:POKE X,175:POKE X+1,50:POKE X+2,0:POKE X+3,60:
POKE X+4,60:POKE X+5,24:POKE X+6,250      <ENTER>
SYSTEM                                     <ENTER>
/20480                                       <ENTER>
```

Listing 10-3. Write circuit diagnostic/BASIC.

This routine will write the value in A, which is incremented from 00 to FF (0 to 255) each time the loop is passed through. The write line will pulse each time the LDD (3C00),A instruction is commanded. The first position on the video screen will flicker, as this is the memory location being written to. Press the Reset button to return from this routine.

## 2. Checking the Read Circuits.

Since each instruction must be fetched, this is only a simple loop:

```
Coding:
18 FE      JR      $
```

Listing 10-4. Read circuit diagnostic/machine.

```
From BASIC:
POKE20480,24:POKE20481,254      <ENTER>
SYSTEM                          <ENTER>
/20480                          <ENTER>
```

Listing 10-5. Read circuit diagnostic/BASIC.

The Read line will pulse four times for every loop through this routine: twice for each instruction fetch, and twice for each refresh action. Press the Reset button to return from this routine.

Listing 10-6. Output circuit diagnostic/machine.

## 3. Checking the Output Circuits.

This diagnostic is very much like that for examining the Write line, except that it triggers the Out line.

```
Coding:
AF      XOR      A
03 FF      OUT      (FF),A
3C      INC      A
18 FB      JR      $-3
```

```
From BASIC:
X=20480:POKEX,176:POKEX+1,211:POKEX+2,255:POKEX+3,60:
POKEX+4,24:POKEX+5,251      <ENTER>
SYSTEM                      <ENTER>
/20480                      <ENTER>
```

Listing 10-7. Output circuit diagnostic/BASIC.

Each time the loop is passed through, the OUT line will be pulsed once, and the data present on the data lines will be incremented. This routine is also very useful because the cassette relay, the cassette data output, and the video screen will all demonstrate activity as the routine is looped through. Press the Reset button to return from this routine.

## 4. Checking the Input Circuits.

This routine is similar to the write routine, but does not include any accumulator changes.

```
Coding:
08 00      IN      A,(00)
18 FC      JR      $-2
```

Listing 10-8. Input circuit diagnostic/machine.

```
From BASIC:
X=20480:POKEX,219:POKEX+1,0:POKEX+2,24:
POKEX+3,252      <ENTER>
SYSTEM          <ENTER>
/20480          <ENTER>
```

Listing 10-9. Input circuit diagnostic/BASIC.

Press the Reset button to return from this routine.

## 5. Checking the HALT Line.

Since the HALT line is gated together with the Reset button, executing HALT should return to ready (or reboot in a disk system).

```
Coding:
76      HALT
```

Listing 10-10. HALT line diagnostic/machine.

```
From BASIC:
POKE20480,118      <ENTER>
SYSTEM            <ENTER>
/20480            <ENTER>
```

Listing 10-11. HALT line diagnostic/BASIC.

Since the machine should return to READY, the Reset button need not be used.

## 6. Checking the Video.

This routine presents a screen full of characters, all identical, and increments through all 256 of the possible characters. With no lower case modification the screen will display three sets of upper case characters, followed by two sets of graphics characters. With a lower case modification, two upper case sets and one lower case, or one of each case plus a set of control characters will be displayed.

```
Coding:
AF      XOR      A
21 00 3C      LO      HL,3C00
11 01 3C      LO      DE,3C01
01 FF 03      LO      BC,03FF
77          LO      (HL),A
E0 80          LDIR
3C          INC      A
F5          PUSH     AF
01 00 C0      LO      BC,C000
C0 60 00      CALL    0060
F1          POP      AF
18 E9          JR      $-210
```

Listing 10-12. Video routine diagnostic/machine.

```

From BASIC:
X=20480:POKE X,175:POKE X+1,33:POKE X+2,0:POKE X+3,60:
POKE X+4,17:POKE X+5,1:POKE X+6,60:POKE X+7,1:
POKE X+8,255:POKE X+9,3:POKE X+10,119:POKE X+11,237:
POKE X+12,176:POKE X+13,60 <ENTER>
POKE X+14,245:POKE X+15,1:POKE X+16,0:POKE X+17,182
POKE X+18,205:POKE X+19,86:POKE X+20,0:POKE X+21,241
POKE X+22,24:POKE X+23,233 <ENTER>
SYSTEM <ENTER>
/20480 <ENTER>

```

Listing 10-13. Video routine diagnostic/BASIC.

Also, the VID\* line (noted on the schematic in the Technical Reference Handbook) should pulse in very noticeable groups as each screen is printed. Use the Reset button to exit this routine.

## 7. Checking the Cassette Output.

This routine calls the byte-output routine, and should write an FF (equal to eight timing pulses and eight data pulses) to port FF (the cassette output).

```

Coding:
3E FF      LD    A,0FF
C0 35 02   CALL 0264
18 F9      JR    $-5

```

Listing 10-14. Cassette output diagnostic/machine.

```

From BASIC:
X=20480:POKE X,62:POKE X+1,255:POKE X+2,205:POKE X+3,100:
POKE X+4,2:POKE X+5,24:POKE X+6,249 <ENTER>
SYSTEM <ENTER>
/20480 <ENTER>

```

Listing 10-15. Cassette output diagnostic/BASIC.

You should be able to measure (or hear) a constant group of pulses output to the cassette player. Note that the cassette player must be in record position, and must be running (the small plug must be removed) because this routine does not turn the cassette machine on. Use the Reset button to exit this routine.

## 8. Checking the Printer.

By writing data to address 37E8, the printer should react by printing that character. The following two routines output the letter 'A' to the printer; the first loops through a delay, outputting about five characters per second. The second waits for a printer handshaking signal.

### Print-and-Delay Routine

```

Coding:
3E 41      LD    A,41
32 E8 37   LD    (37E8),A
01 00 30   LD    00,3000
C0 60 00   CALL 0060
18 F3      JR    $-110

```

Listing 10-16. Printer diagnostic I/machine.

```

From BASIC:
X=20480:POKE X,62:POKE X+1,65:POKE X+2,50:POKE X+3,232:
POKE X+4,55:POKE X+5,1:POKE X+6,0:POKE X+7,48:
POKE X+8,205:POKE X+9,98:POKE X+10,0:POKE X+11,24:
POKE X+12,243 <ENTER>
SYSTEM <ENTER>
/20480 <ENTER>

```

Listing 10-17. Printer diagnostic I/BASIC.

### Print-and-Wait Routine

```

Coding:
3E 41      LD    A,41
21 E8 37   LD    HL,37E8
77         LD    (HL),A
C8 76      BIT   6,(HL)
20 FE      JR    NZ,$
18 F4      JR    $-100

```

Listing 10-18. Printer diagnostic II/machine.

```

From BASIC:
X=20480:POKE X,62:POKE X+1,65:POKE X+2,33:POKE X+3,232:
POKE X+4,55:POKE X+5,119:POKE X+6,203:POKE X+7,118:
POKE X+8,32:POKE X+9,254:POKE X+10,24:POKE X+11,244 <ENTER>
SYSTEM <ENTER>
/20480 <ENTER>

```

Listing 10-19. Printer diagnostic II/BASIC.

## 9. Checking the Interrupt Line

For this, of course, interrupt hardware must be attached to the system. In the case of the expansion box, the interrupt flip-flop must be cleared (see Supplement to Chapter 4). This routine performs that function. It is identical to the first service routine presented in the Supplement to Chapter 4, and so only the BASIC coding is presented.

From BASIC, attempt to use this program instead of direct POKEs, because there are so many values:

```

10 FOR X = 20480 TO 20926 : READ A : POKE X,A : NEXT
20 STOP
30 DATA 243,62,195,50,18,64,33,20,50,34,18,64
40 DATA 33,25,26,228,237,86,251,205
50 DATA 243,245,229,213,187,58,236,55
60 DATA 58,224,55,33,17,1,17,37,62,1,26,0
70 DATA 237,178,193,209,225,251,201

SYSTEM <ENTER>
/20480 <ENTER>

```

Listing 10-20. Interrupt line diagnostic/BASIC.

This routine, as described, returns to BASIC command level and displays a RADIO SHACK LEVEL II BASIC message continuously on the screen.

## 10. Checking Speed Modifications

Among the possible high speed failures are miswirings, ROMs which are too slow, and RAMs which are too slow. Combinations of these can make diagnosing a locked-up computer very frustrating.

The two routines below test, respectively, RAM calling ROM, and RAM alone. The RAM routine may be moved to higher memory for testing that area; a version at 5000 and B000 are provided.

Coding:

### RAM-Resident Version – Origin is at 5000

```

32 60 30      LO      (3060),A
3C            INC     A
F5            PUSH    AF
AF            XOR     A
03 FE        OUT     (FE),A
F1            POP     AF
32 61 30      LO      (3061),A
F5            PUSH    AF
3E FF        LO      A,FF
03 FE        OUT     (FE),A
01 00 10     LO      BC,1000
* 0B         DEC     BC
* 7B         LO      A,B
* B1         OR      C
* 20 FB      DJNZ    FB
F1            POP     AF
% C3 00 50    JP      5000

```

#### ROM-Resident Version

Remove lines marked (\*) and replace with:

```

C0 60 00      CALL    0060

```

#### High RAM-Resident Version

Origin is at B000; replace line marked (%) with:

```

C3 00 B0      JP      B000

```

For the ROM-RAM version, replace the last group of POKEs with:

```

POKE+21,205:POKE+22,96:POKE+23,0:POKE+24,241:
POKE+25,195:POKE+26,0:POKE+27,80      <ENTER>

```

For the high RAM version, let X = -20480, and replace the last value POKEd (in either configuration) with 176.

These routines will present, just below the center of the screen, a pair of characters. The first of these is printed in the machine's normal configuration (if OUT 254,0 sets your machine to normal). The second character is printed with the effect of OUT 254,255, which should turn all modifications on. Any failure in these routines should point to the specific area of breakdown.

Listing 10-21. High-speed modification diagnostic machine.

### RAM/ROM Version

Remove lines marked (\*) and replace with:

### High RAM-Resident Version

Origin is at B000; replace line marked with percent sign with:

Since speed changes can affect the operation of BASIC, a version with POKE may be useless; if not, however, here is the RAM-resident version:

```

X=-20480:POKE,50:POKE+1,96:POKE+2,61:POKE+3,60:
POKE+4,245:POKE+5,175:POKE+6,211:POKE+7,254:
POKE+8,241:POKE+9,50:POKE+10,97:POKE+11,61:
POKE+12,60:POKE+13,245:POKE+14,62:POKE+15,255:
POKE+16,211:POKE+17,254:POKE+18,1:POKE+19,0:
POKE+20,16      <ENTER>
POKE+21,11:POKE+22,120:POKE+23,177:POKE+24,32:
POKE+25,251:POKE+26,241:POKE+27,195:POKE+28,0:
POKE+29,80      <ENTER>

```

Listing 10-22. High-speed modification diagnostic BASIC.

# NOTES

---

# 11

---

## 111 CURES FOR THE COMMON CRASH

Because TRS-80 computers are sold at the 'appliance' level, salespeople and users often forget that they are indeed computers, prone to all the problems that larger computers face. As a user of a personal computer, you become your own service representative, your own diagnostician, and your own repair agency. This is even more true if you have a custom TRS-80.

This chapter is dedicated to you, the user, as service rep, diagnostic engineer, and repair person.

### I. SOFTWARE

*Symptoms of software problems: all error messages, system lockup and crash, just about everything that doesn't snap, crackle or pop.*

#### 1. Correct the program – the greatest cause of crashes.

By far the largest cause of computer 'crashes' is the software. Mishandling of POKE, PEEK and VARPTR are most common as well as improper input/output routines, and insufficient error-checking.

If crashes seem to occur only in specific programs, check these for errors before turning to hardware cures, although a 'stuck bit' may always turn up a consistent error.

#### 2. Set memory size correctly for machine language or hybrid programs.

Memory size is a boundary above which BASIC is prohibited. BASIC is not merely the source program, but consists of variable and string storage and the BASIC stack. Some of this uses high memory, and when memory size is not set as specified, these variables and stack information can run into the high-memory machine language program.

#### 3. Wait out or re-write long string searches and sorts.

The memory allocation method for strings has made 'garbage collection' techniques necessary. When sorting is complex and the number of strings is large, this process can take from a few minutes to an hour, during which the computer seems locked up. One cure is clearing as much string space as possible; when all variables have been defined, break and PRINT MEM. To the program, add a line such as CLEAR MEM-N, where N is about 100 bytes more than the difference between the

program with all variables defined and your total memory; subtract about 20 bytes from the CLEAR statement for each nested FOR-NEXT loop and GOSUB. Refer to 'BASIC Faster and Better' and other books for techniques of rewriting string handling using variable pointers, and otherwise using memory economically.

#### 4. Read the manuals.

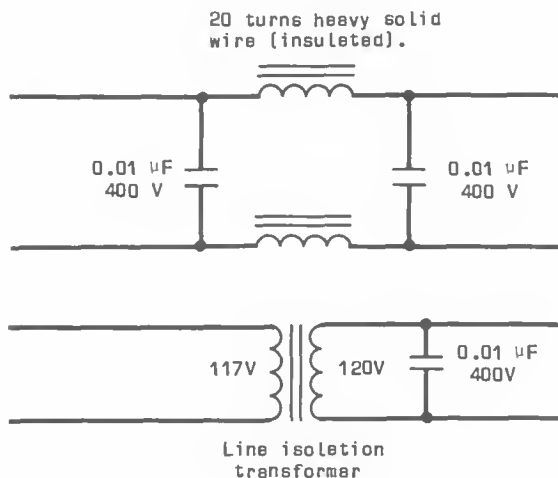
Most bugs are called 'features' by program authors. Discover these features in the program documentation.

### I. POWER

*Problems Caused by Poor Power: system reboot, unexpected syntax errors, speckles on screen, shrinking screen, unexpected disk or Stringy-Floppy startup, total lockup of system.*

#### 5. Add a line filter or power supply monitor.

'Clean' power enhances system reliability. The simplest help is a line filter such as that sold by Radio Shack. More severe power problems can be ameliorated by using power supply monitors such as Topax or Mayday. Isolating other appliances (power tools, washers, refrigerators, toasters, water pumps, fluorescent lamps, etc.) from your computer's outlet will enhance reliable operation. You can even build some very simple filters yourself:



#### 6. Use 3-wire grounded circuitry for all equipment.

3-wire circuitry is electrically quieter and safer. Some equipment, particularly printers, have damaged computers when used in an ungrounded or improperly-grounded environment. If you have an older, 2-wire system, test for floating voltages using a neon test light, available from hardware stores. Reverse the plugs in the socket until all voltages disappear.

#### 7. Stop strong power drains (granite sheds, arc welders).

Strong power drains are visible as brief but distinct shrinking and dimming of the video. Cassette and disk I/O problems are most likely, but power drains can sometimes result in program failure. A high quality power supply monitor can help. Also, if you don't live in an industrial area, your power company or zoning commission might be able to assist.

#### 8. Fuse in the power supply might be going or gone.

All separate TRS-80 power supplies contain fuses in order to receive an Underwriters Laboratories approval. Earlier supplies are sealed, but the seal can be broken by forcing a screwdriver around the joint at the bottom of the supply and gradually popping off the top section. Later supplies have screws under the rubber feet. Fuses are cheap. Don't test the fuse, replace it. It will need to be resoldered since these fuses are not in sockets.

#### 9. Replace or resolder faulty power cables at the connector.

The power connector is not molded in place, and after a year of abuse, the soldered connections can break inside. Lift the tabs on the plastic sheath and pull it back. The connector can be resoldered.

If the power remains intermittent, the problem may be at the point where a band of metal is crimped around the cable. Loosen it, cut the cable shorter, and resolder. The 5-pin DIN plug is sold by Radio Shack if you want to replace it completely.

#### **10. Install a lightning arrestor in spite of the phone company.**

If you have a direct-connect modem, lightning strikes may be damaging your system, or at least affecting its performance. For best protection, always unplug the system from the phone and power lines when a storm is in the area. You can also obtain a TII lightning arrestor from Datadyne, 450 Seventh Avenue, NY 10001.

## **II. EDGE CONNECTORS**

*Symptoms of edge connector problems: system reboot, unexpected syntax and line number errors, loss of the end of a program or text file, system lockup, return to READY before program end.*

#### **11. Clean edge connectors by erasing them.**

Since the solder-coated edge connectors are prone to corrosion, they must be cleaned regularly. The simplest methods are vigorous rubbing with a piece of coarse paper like a dollar bill, or erasing with pink pearl (good) or white plastic (best) erasers.

#### **12. Spray edge connectors with contact cleaner and swab with cotton.**

Once you have cleaned the edge connectors thoroughly, a weekly application of a small amount of spray contact cleaner (such as Radio Shack 64-2320), followed by rubbing with a cotton swab, will keep the contacts in good shape.

#### **13. Emery-paper edge connectors for really bad corrosion.**

For really bad corrosion or scoring from continuous insertion and removal of connectors, the finest grade of emery paper or cloth can be used. Use wet-or-dry paper soaked in contact cleaner or diluted isopropyl alcohol, and rub smoothly along the connectors. Follow up with more contact cleaner and cotton swabbing. Rub just enough to bring up a shine, and do not use coarse emery paper. Never sand down to the copper traces.

#### **14. Silver solder the edge connectors with Silver-It.**

Corrosion can be reduced to a minimum while maintaining the original physical size and shape of the edge connectors by obtaining a Silver-It kit from Fuller Products, Grand Prairie, Texas. This is a process that must be completed with great care, but will result in connectors that (except in chemically violent atmospheres in major cities) need virtually no cleaning. For Radio Shack repairs, this process looks like no modifications have been made.

#### **15. Gold plate the edge connectors.**

Most difficult of the solutions is gold plating, which is also a poisonous process. However, when it is complete, the result will be connectors of the original physical size with a corrosion-free gold coating. Refer to 80 Microcomputing, December 1981, for full details.

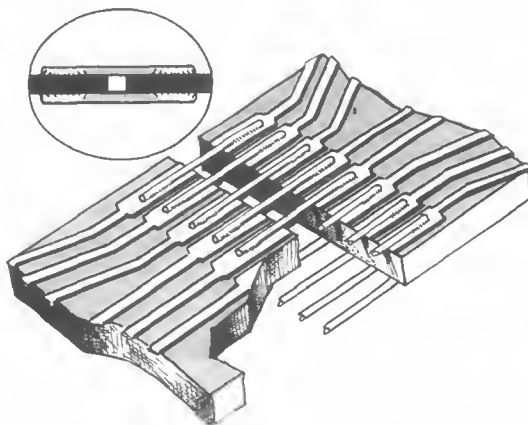
#### **16. Replace the connectors with gold plugs from EAP.**

Easier than silvering or gold plating and less annoying than regular cleaning are replacement gold edge connectors (Gold Plug 80) sold by EAP Products, Box 14, Keller, Texas. Because the connectors are made by Kel/Am, they do not mate properly with AP Products connectors. Peri-

peripherals using AP connectors can be easily changed to use the T&B/Ansley connector sold by Radio Shack. Also, the Gold Plug 80 connectors will protrude from the keyboard unit and expansion box somewhat less than an inch.

**17. Solder all the connectors into one box and one board.**

Model III cases can be purchased as replacement parts through Radio Shack's National Parts distribution system. If you wish to have a one-piece system, this can also cure the edge connector problem. Heavy pieces of copper wire can be soldered to the keyboard and expansion edge connectors, effectively creating a single large board. Don't solder a cable in place and then put one board on top of the other, because electronic noise will be a problem.



**18. Both unbuffered and buffered cables are inserted only one way.**

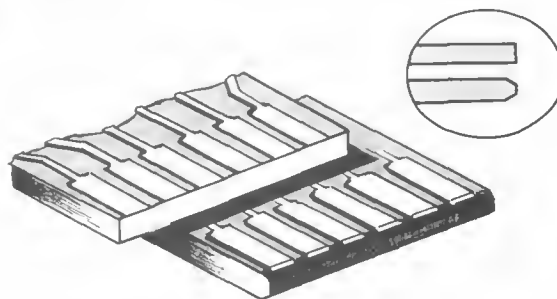
The buffered cable is marked clearly as to which end connects to the keyboard and which end to the expansion interface. However, the unbuffered cable is also directional because it contains a shield which should be connected to ground. If the cable is reversed, the large shield will be connected to a system reset signal — a sure troublemaker. Examine the cable for a fine copper wire sliding out from between the grey plastic 'sandwich'. It should be on the left side of the cable when the cable is attached, with the smooth face of the cable toward the top.

**19. Cables not inserted all the way or crooked on connector.**

Interconnect cables, particularly AP Products connectors, need to be fully inserted to make electrical contact. Although the expansion cable is not usually inserted partly or crooked, it is easier to do with the printer and disk cables, which are somewhat out of sight. Also, Kel/Am connectors have a limited life — about 50 insertions and removals before the contacts bend inside the connector. Examine the end — all contacts should be even.

**20. File carefully to cure a tight fit on the edge connector.**

Most Model I TRS-80's contain printed circuit board slightly thicker than that normally used with edge connectors. Although the cable will go on, it has to be forced. To avoid this, smooth the very edge of the circuit board with a fine file, to round the edge. File back not more than 1/8 inch:



### III. MEMORY

*Symptoms of memory problems: erratic operation, unexpected syntax, unidentified-line, next-without-for and subscript-out-of-range errors, incorrectly reported memory on power-up, program lock-up.*

**21. Replace slow memory with faster chips, but check speeds.**

All dynamic memory has a window during which data is valid for the CPU. Refer to Chapter 4 for a table of memory speeds for keyboard unit and expansion interface. Also, note that memories can deteriorate with age, and poorer access time is one of the first signs of age.

## 22. Replace bad memory with good memory, using a memory test.

A memory test will reveal memory chips which are genuinely bad, identifying both their address and the bit (chip) involved. Remember that addresses **4000-7FFF** are in the keyboard, and the expansion interface contains two banks, **8000-BFFF** and **C000-FFFF**.

## 23. Replace older memory with newer (post-1979) memory.

Early dynamic memories were plagued with the occasional 'soft error' — a program crash not caused by any permanent hardware affliction. The error was finally identified as a slow emission of alpha particles from the substrate (base) of the memory circuitry inside the chip. These particles would strike a memory cell, changing it. The occurrence was unpredictable. The best cure for this dilemma is the replacement of all such memories. Date codes are stamped on the chip in year/week format, such as 7851 (end of December 1978); use memories with an 8001 or later date code.

## 24. Increase access with a modification to Z69.

If the problem seems to be slow or aging memory, the Z69 modification can help. Cut trace from Z69 pin 5. Attach Z69 pin 12 either to Z69 pin 10 or to Z69 pin 13.

## 25. Install a buffered cable according to modified Radio Shack instructions.

When memory failure seems to occur as more items are added to the system via the expansion interface, then (in earlier units) a buffered cable is called for. It provides additional fan-out (drive) capabilities to the keyboard unit. This item is available from Radio Shack at no charge

to owners of early expansion interfaces. You can identify the earlier interfaces because the 32K memory runs from back to front; in newer boxes it runs along the back. Refer to Chapter 4 for installation instructions.

## 26. Install the twisted-pair DIN plug modification.

In addition to the buffered cable, some earlier interfaces may require the 'twisted pair' modification. This is a 6-wire DIN cable combination to carry memory select signals RAS, CAS and MUX. Refer to Chapter 4 for installation instructions.

## 27. Modify twisted pair resistors upward to 470/680 ohm pairs.

If the installation of the twisted pair mod deteriorates rather than ameliorates operation, change the resistor values specified by Radio Shack to 470 ohms to ground, and 680 ohms to 5 volts. Both are originally 220 ohms.

## 28. Straighten bent pins under socketed ICs.

Memory chips and the CPU were the only chips socketed in earlier machines. However, various runs of the computer had the character generator, line buffers, and so forth, in sockets. Evidence of bent pins is a machine which acts up when given a light physical shock — such as hard typing. Lift each IC and check for bent pins; be sure not to bend any pins when putting it back! If your machine has had a recent trip to salt-water environments, the problem will be more severe; see #104.

## IV. FIRMWARE

*Symptoms of firmware problems: incorrectly read data, equality failures in IF-THEN statements.*

### **29. POKE 16553, 255 to correct READ/DATA error.**

In Version 1 of the Level II ROM, under certain conditions the same data would be read over and over, with the data pointer not being stepped through memory. The most common fix is POKE 16553,255 added as the first line of any program. Also, any INPUT statement before the reading of data begins will take care of properly stepping the pointer.

### **30. There are floating point accuracy errors, such as X-Y<> X-Y%.**

Because of the way real numbers are handled digitally, there is a very small numeric residue left after some types of calculations. Where IF-THEN statements don't seem to work when you know they should, break into the program and print the offending variables. Chances are you will see that a value, instead of being 20 as you expected, is actually 20.00001 as a result of residual binary information. Use integers where you can for such tests; see the Level II manual for details.

## **V. RS-232**

*Symptoms of RS-232 problems: incorrectly received or transmitted characters, system crashing or lockup when not using RS-232, electronic failure.*

### **31. Place bar across RS-232 to keep it in place.**

Heat buildup and general stress in the expansion box will cause a warping of the RS-232 board, lifting some contacts above the connection pins, or making the unit vibration sensitive. By using longer mounting screws and a heavy insulated metal bar, the warp can be prevented. You may have one made at a local metal shop, then cover it with 'heat-shrink' tubing sold by Radio Shack. Recently, some sources have been making such bars available; look for ads.

### **32. Clean RS-232 contacts vigorously with cotton cloth.**

These contacts too are solder-plated, and prone to corrosion. But they are also very thin, and only a clean cotton cloth - together with vigorous rubbing (and perhaps a little contact cleaner) should be used. Make the contacts shiny.

### **33. Reseat the RS232 board, checking for bent pins.**

The RS-232 board contacts are 1/20 inch from center to center, exactly half the size of the edge connectors. Bent pins on the expansion interface RS-232 connector can cause shorts which will affect not only the operation of the RS-232 system, but the computer as a whole. Reseat the board, checking under strong light that all the connecting pins are straight and contact properly.

### **34. COM/TERM on RS-232 must be in correct position.**

Recheck the RS-232 manual, and make sure that the Communications/Terminal switch is in the correct position for the software you are using.

## **VI. DISK**

*Symptoms of disk problems: lost data, hang-up during disk access, rattling and banging of disk mechanism.*

### **35. Use better disks, the best you can get if data is critical.**

If you've spent an hour entering data, you've paid for the best diskette you can buy. Purchasing cheap disks is false economy.

### **36. Install a data separator, doubler, or other separation.**

Data is stored on disk as a continuous stream of clock pulses separating data pulses. As a matter of economy, Radio

Shack chose to use 'internal' separation — that is, using the disc controller integrated circuit to distinguish between clock and data pulses. The disc controller's manufacturer does not recommend this method; therefore, a piggyback data separator (sold by several sources) reinstates the proper electronic design. Similarly, a double-density controller contains the essential data separation. If you obtain a 1771B-01 data sheet from Western Digital Corporation, you can build your own data separator.

### **37. Align the disk drive read/write head professionally.**

When a single-drive system is in use, misalignment of a drive head might not be noticed. However, in a multi-drive system, one drive may produce an unusual number of re-seeks or error messages. If you suspect misalignment of the head, don't attempt to service it yourself. You can obtain an alignment diskette (a good one is manufactured by Dysan) to confirm your suspicions, but professional service is called for in this case.

### **38. Clean the disk drive read/write head with cleaning disks.**

When a large number of disks are in use, especially inexpensive ones, tiny bits of oxide are shed onto the drive read/write head. Since this is not immediately visible, it's good practice to obtain a disk cleaning kit (Scotch and others) and use it weekly.

### **39. Erase-clean connections to the disk drive.**

The edge connector to the disk drive is afflicted with the same corrosion problem as other edge connectors. The solutions are the same: cleaning, plating, and replacement with gold plugs. See under edge connectors (#11 above) for more information.

### **40. Replace 74LS38/LS16 clock selector ICs and socket them.**

The cable to disk drives is long and must be driven by higher-current integrated circuits, type 74LS38 (74LS16 in the newer expansion box). These can often break down under continuous use and in situations where the drives are often plugged in incorrectly. The most evident symptom is failure during formatting or backup, because the stepping signal (low to move outward, high to move inward) deteriorates and missteps the read/write head. Remove these chips, and solder in sockets. Then keep a small stock handy so they can be replaced when these symptoms show up.

### **41. Lubricate disk drive and rails sparingly with silicones.**

Many disk drives squeal and clatter. There is no need to risk mechanical failure. Remove the cover and lubricate the motor bearings and the guide rails; use only a very light grade silicone lubricant, and wipe any extra lubricant off once it has spread across the area that needs it. Refer to Exclusive Oracle, 80 Microcomputing, January 1982, for details.

### **42. Keep the disk door closed till it stops to protect loaded heads.**

Your disk drive may contain 'loaded' heads, which means that the door mechanics do not lift the head away from the disk when the door is opened. If you remove or insert a disk before the select light goes out, you may damage the head assembly. You can tell if your drive has loaded heads from the documentation, or by listening for a 'clack' when the drive is selected, and a second 'clack' when the select light goes out. In any case, it is good practice to leave the disk in place in any drive while the motor is spinning.

#### **43. Put disks in correct keyed cable position.**

Disk drive selection (drive 0 to 3) can be made in two ways: the drive itself may be programmed with internal jumpers, or the cable may have teeth pulled to eliminate the unwanted select signals. If your drive programming and missing teeth do not match, then the drive will appear dead when selection is attempted. Make sure your drives are marked 0, 1, 2 and 3, and that you match them correctly to the cable.

#### **44. Disk cable must be right side up.**

If your disk drive keeps running and doesn't otherwise work, the drive is probably plugged into the cable upside down. Try reversing the connection. Since many drives have cables protruding out the back, and the physical position of these cables is not standard, the plug-in may look correct with respect to the other drives, but be backwards.

#### **45. Disk cable must be fully installed inside case.**

The cable protruding from the back of some brands of disk drives is usually not hard-wired, but simply an extension cable from an internal edge card. Under the weight of the long multi-drive cable, or simply from regular moving and unplugging, this internal connector can come loose, resulting in erratic operation. Remove the case top and re-insert the extension cable. A piece of strong plastic packing tape can keep this cable from shifting.

#### **46. Insert the disk correctly into the drive.**

This is not as obvious as it seems. If you mix different brand drives on your system, you may discover some in which the disk must be inserted with the write-protect notch pointing down rather than up.

Usually the write-protect notch points toward the side of the drive where the select light is mounted, but even that is not standard. Be especially careful about this when using an unfamiliar system, as a hangup during disk access can be fatal to data in memory.

#### **47. Remove or pad sources of vibration nearby disks.**

If you have lived or worked in an area for some time, you have probably blocked out sources of vibration, such as heavy equipment, trains, etc. However, this kind of vibration can ruin a disk write or read. The simplest solution is padding: a layer of heavy cloth, a layer of cork, a layer of wood or metal. If you have occasional inexplicable read/write errors, tune your senses to the environment.

#### **48. Update the DOS, especially if it's an early version of TRSDOS.**

Most of the bugs and inconveniences of early versions of TRSDOS have been corrected either by Radio Shack, or by other software houses who have created new disk operating systems. To avoid frustrating errors, update your DOS.

#### **49. Keep disks clean, unbent, and store them straight up in cases.**

Though this may seem obvious, there are hidden causes of dirt: smoking, heavy dust or other airborne particles, air freshener sprays, animals, etc. Bending can be caused by storing disks sideways, keeping them in a car window, or inserting them hastily into the drive. Don't drink soda nearby — unless you put your disks right back in their sleeves (which you should do); the bursting bubbles of carbonation can carry sugar residue to the disk surface, damaging the recorded data and abrading the disk head.

## VII. TAPE

*Symptoms of tape problems: loading impossible, hangup during load, tape won't go on or off, tape won't record.*

### 50. Align the tape recorder head by drilling a hole.

Misalignment is singly the largest tape problem. Put the recorder in playback mode with no cassette in place, and shine a bright light so you can see the Phillips alignment screw to the left of the tape head. Drill a hole directly above it. Align the head by popping in a good commercial music tape, and turning the screw until the sound is at its brightest. Use this for standard recording and playback, but readjust for any commercial tapes that don't sound 'bright'.

### 51. Clean the tape recorder head with isopropyl alcohol only.

An oxide buildup is common on all tape recorders. Using head cleaner or isopropyl alcohol (not acetone!), swab the tape head and other metal parts which exhibit brown oxide caking. This will prevent scratching or scraping of the tape surface, as well as ensure good contact with the tape head.

### 52. Demagnetize the tape recorder head with proper devices.

The high frequencies are the most crucial element in good tape loading, and a magnetized tape head erases some of these high frequencies every time the tape is played. Pick up a cassette tape head demagnetizer, either a plug-in type or the kind packaged in a cassette case, and demagnetize the cassette player at least monthly.

### 53. Use better tape but not the very best audio stuff.

Good tape will always give good loads. Avoid inexpensive tapes like Certron, Concertape, and questionable depart-

ment store house brands. Radio Shack red-label Realistic tape is just fine, as is most any good audio tape. Very high quality tapes (chromium dioxide, metal, etc.) are not necessary, except for archival backups. Digital tapes from Microsette are only about \$.65 and are sold in handy lengths (C-10 and C-20).

### 54. Modify the CTR-80 with a diode to prevent head glitches.

If the stop button was pressed during loading, a head field collapse in early CTR-80 tape recorders would put glitches on program tapes. Radio Shack provides a free modification for this problem. If you wish to do it yourself, a small silicon diode (such as type 1N4148) can be connected across the tape head contacts.

### 55. Replace cassette relay to prevent sticking.

The current drain of the CTR-41 tape recorder, and many non-Radio Shack recorders, is too high for the relay contacts in the keyboard unit. This causes it to stick closed, keeping the tape recorder running when it should not. There are two options: change tape recorders, or change relays. The relay change is permanent, and a new unit can be obtained from Lab Service, Inc., in Hustisford, WI.

### 56. Use CTR-80 to prevent cassette relay burnout.

If you don't want the trouble of replacing a relay, use a CTR-80 tape recorder. It also prevents wear and tear on the cables, because rewind and fast-forward can be used without pulling the motor-control plug.

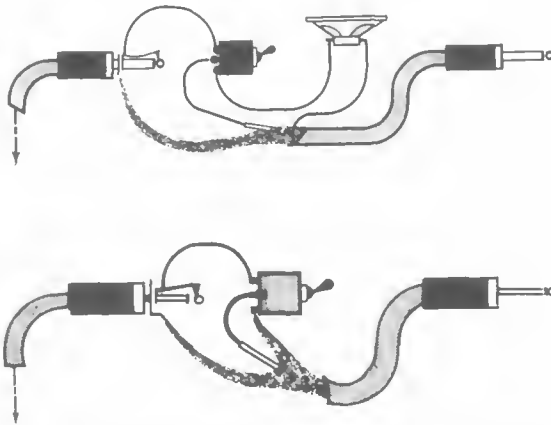
### 57. Clean the capstan and pinch roller with isopropyl alcohol.

As the tape recorder sees a lot of use, two things will happen to the rubber pinch roller: oxide buildup and glazing. Oxide buildup is similar to that on the tape head,

but on the roller it can cause tape to slip out of place, creasing it. Glazing is a shininess of the rubber surface, also causing tape slip and speed variations. Both can be cured with the liberal application of isopropyl alcohol, acetone, or tape roller cleaner. Hard-glazed rollers need to be brought down with fine emery paper.

**58. Add a switch box to the cable assembly for hand operation and sound.**

A very useful addition to your tape system is a small switch box containing a speaker; headphone speakers are best. With a simple two-switch box you can listen to programs being saved, keep the motor running, and stop it when you wish:



**59. Get the XRX-II cassette modification.**

Radio Shack created a special cassette system modification, which is available at no charge. It improves 500-baud loading, but consists of a 500-baud window — which means using cassettes running at any other baud rate (hardware or software based) is impossible. However, the modification can be switched out if you wish to have the advantages of good 500-baud loading and not lose the option of other speeds:

**60. Ground loop hum in cassette system is cured by breaking the loop.**

Some loading problems can be attributed to hum during program saves. This is due to a 'ground loop' created by the input and

output cables. Pull back the plastic sheath on the computer end of the cassette cable, and cut either one — but not both — of the two shields going to ground (pin 2, the center pin). This will kill ground loop hum.

**61. Replace or resolder a faulty cassette cable at the connector.**

If you often pull the cassette cables out of the recorder, you may break the internal wiring. This will be evident as a crackling sound during program saves, or partial good loads from tapes which once loaded perfectly. You can replace the connectors with mini and submini plugs, or you can obtain a complete new cable.

**62. Replace the dual cassette relay or driver IC.**

If the dual-cassette system from the expansion interface ceases to function, either the relay or its associated driver integrated circuit (Z41) may be bad. You can test the operation with:

```
1 A=14308:POKEA,0:FORX=1TO9:NEXT
:POKEA,1:FORX=1TO9:NEXT:GOTO1
```

The relay in the expansion box should clack rapidly.

**63. Cassette cables must go to correct expansion box positions.**

If one cassette seems to load and the other doesn't, then check that the cassette cables are properly installed. The order (looking from the back) is not particularly logical: Common Cable, Cassette #1, Cassette #0.

## VIII. HARDWARE

*Symptoms of hardware problems: loss of power, changing of memory contents, keyboard lockup, no apparent expansion interface memory, miscellaneous woes.*

**64. Tighten the power transistor screw and resolder screw head.**

The keyboard unit's power transistor (Q2) was screwed into place, with the screw acting as an electrical connection. This screw can corrode, resulting in erratic operation and frequent complete system crashes. Remove, clean and tighten this screw.

**65. Terminate data lines either up or down, but only once.**

The eight data lines are flying free in the TRS-80 system. For reliable operation, they should be terminated with resistors — but only once. Since some peripherals contain terminating resistors, the best idea is to obtain an edge connector, and solder resistors to it. Eight resistors (680 ohms, 1/4 watt) go from data lines to ground, eight resistors (470 ohms, 1/4 watt) go from data lines to 5 volts.

**66. Add two capacitors to RAM bank the expansion interface.**

The absence of two capacitors from the expansion interface has never been explained. There was a place for them, and their inclusion improves operation. Install two electrolytic capacitors (10 mF, 16V) in the positions marked C54 and C55 in the two rows of memory in the expansion interface.

**67. Replace the CPU with a Z80A for high speed, or a Z80B for best operation.**

If you are running your computer at high speed (100% increase or greater), you are exceeding the formal specifications for the Z80 processor. Although many Z80s can run at that higher speed, certain complex operations (stack operations, for example) can fail. Replacing the chip with a Z80A (4MHz version) or Z80B (6MHz version) will enhance reliability.

**68. Replace memory for high speed, but observe DDU wiring changes.**

Replacing expansion interface memory for high speed operation may not always help run the machine reliably. Newer expansion boxes contain a digital delay unit for memory selection, which has a fixed access time. Refer to Chapter 4 for wiring changes to the DDU for high speed.

**69. Check power supply voltages with a calibrated meter.**

Since the miniature voltage adjustment controls were not lacquered, vibration can cause them to move, resulting in the voltages being out of calibration. Units showing as little as 3.2 volts on the 5-volt line have been seen. Obtain a calibrated meter (not a \$10 off-the-shelf number) and adjust the voltages — 12 volts first, then 5 volts. These are R10 (12 volts) and R5 (5 volts) in the keyboard unit.

**70. Increase C48 in the expansion interface to keep the disk going.**

A discouraging aspect of some disk operating systems is that they cannot recover from a disk drive which has 'timed out' — the motor has stopped. Some DOSes contain a Shift/Break option, but this does not always function, depending on when the disk timed out. A better solution is to increase the value of C48 (C62 in the newer expansion box) from 33 mF to 47 mF or 68 mF. Use a bead tantalum capacitor, not an aluminum electrolytic, if possible.

**71. Add or remove disk termination resistors.**

In order for disk selection to take place properly, termination resistors are required. Normally, these should only be installed in the furthest drive on the cable. However, as people trade or sell units, or purchase them from a variety of manufac-

turers, the number of termination resistors may vary. These can be found by removing the disk drive cover; they are a red, white or blue integrated-circuit sized package, usually near the edge connector. Make sure only one set of resistors is in place, no matter how many drives are in the system.

## **72. Make the reset modification to the expansion interface.**

It seems the option to turn off a disk system should have been provided with the expansion interface. Ironically, other manufacturers have followed the original unresettable design. You can get out of many program hangups by pressing reset in a Level II keyboard only — so why not with an expansion box attached? Refer to Chapter 4 for details.

## **73. Change LNW termination resistors to 470/680 pairs.**

LNW expansion systems have placed very low termination resistor values in their boards. If too many peripherals (or any peripherals with their own termination) are added to the system, computer lockup will probably occur. For reliability, these values should be changed to 680 ohms to ground, and 470 ohms to 5 volts.

## **74. Check the DDU in the newer expansion interface.**

Sudden failure of memory in new expansion interfaces can almost always be traced to failure in the digital delay unit (DDU), marked Z37. This unit should be slightly warm — neither hot nor cold — to the touch. Since replacement DDU's are about \$20, get some help if you're not sure whether the DDU is bad.

## **75. Make sure the ROM cable is okay, neither pulled from its sockets, nor with bent pins.**

If the computer crashes with an occasional screenful of garbage, or patterns of @9@9,

@A@A, etc., or it fills the screen from the bottom left with A A A , then suspect a flukey Level II ROM cable, or a cracked DIP shunt (see #7 below). Make sure the cable is fully installed. No pins should be bent or broken on the cable; replace it with another 24-pin jumper cable if any pins are bent or broken. (If they're bent, chances are they'll be broken when you try to bend them back).

## **76. Cold solder joints, splashes, balls, etc., can be anywhere.**

This is the worst problem. TRS-80 computer boards are soldered mechanically, and residual solder bits are cleaned away. However, a few balls, splashes or hairs of solder may remain, breaking loose after the vibration of a year of use to cause trouble. Remove all cables and shake out the computer; small solder bits may drop out of the case. Also, broken traces can occur, particularly where any scratches in the green solder mask have occurred. This might be a 'professional help' category.

## **77. Check DIP shunts for correct or accidental breaks.**

Another cause of @9@9, @A@A, @S@S, or the moving A A A A are DIP shunts. Only certain pins (see Chapter 4) should be broken in the DIP shunts (Z3 and Z72 in the keyboard unit). DIP shunts may have hairline cracks; remove them and check with an ohmmeter to be sure. You can replace DIP shunts with DIP switches or ordinary staples.

## **78. Reset problems are a bad CPU and related capacitor, no disk system in use, or are program-caused.**

Users who recently install an expansion interface often forget that the reset button no longer works as it used to, and this is particularly a problem if no disk drives are attached (see #72 above). However, there are three other reasons for reset failure, even if disks are in place: the reset

pin of the Z80 CPU is bad (meaning replace it), or the associated charging capacitor and bleeder resistor (R47 and C42) are bad (replace them), or the program attempts to use the machine language HALT instruction (see Chapter 3).

## IX. KEYBOARD

*Symptoms of keyboard problems: key-bounce, keys not working, multiple different characters, continuous scrolling ?SN or ?S errors on screen, constant repeating memory size question followed by odd characters.*

### 79. Clean keyboard contacts — old one only — not ALPS.

The most straightforward cure for keybounce is cleaning the keyboard. On the bounce-prone keyboards (see photos in Chapter 4), the keycaps lift off. The keys can be cleaned with dry air, spray cleaner (works well but requires cleaning more often), or filled with silicone grease (some people say this improves the feel of the keyboard). The latter technique reduces bounce from vibration as well.

### 80. Use KBFIX and other software instead of a new keyboard.

A debounce program — automatic on most disk systems — can be loaded at the beginning of each session. Radio Shack's KBFIX is clumsy to get loaded, but works. The debounce routine presented in Chapter 3 works well. Note that new ROMs ('R/S L2 BASIC') contain a debounce routine, and a second routine added to this can result in very slow key response.

### 81. Get ALPS keyboard, which is the debounced keyboard.

If you can afford the \$75 replacement cost, a Radio Shack Hall-effect ('ALPS') keyboard will permanently take care of bounce problems. If you switch often between BASIC and machine language

programs, play games, or have a variety of disk operating systems or text editing programs, the ALPS keyboard may be the best solution.

### 82. Replace 74LS05s in the keyboard if multiple or unusual characters appear.

If multiple or unusual combinations of characters appear unexpectedly on the screen, or if characters begin to repeat by themselves, then the 74LS05s on the keyboard baseplate are probably bad. Replace them both, in sockets.

### 83. Replace or repair the keyboard cable if odd stuff shows up.

Another cause of odd character combinations — usually this like 'FIAQ9' or some such when a single or pair of letters is pressed — is a cracked or intermittent keyboard cable. Replace it; see Chapter 10.

## X. VIDEO

*Symptoms of video problems: no video, dim video, screen tearing or twitching, blurred screen, screen glitches.*

### 84. Add a buffer stage or resistor to cure video tear.

Lowering the value of R14 in the video monitor will reduce the 'tearing' present when large blocks of graphics are displayed, especially using reverse video. See Chapter 4 for details. The other option is to obtain a video buffer circuit from Archbold Electronics.

### 85. Add a deglitch modification for a prettier screen.

The video 'hash' created when graphics are being drawn is caused by a conflict between the relatively independent video display circuitry and the need of the CPU to access video memory. Add the deglitch modification for a clearer screen, presented in 80 Microcomputing, Feb. 1982.

### 86. Get a new character generator, with descenders.

Upper case characters on the TRS-80 have always been consistent, but lower case characters can be displayed either with flying letters (g, p, q and y) or with descenders. The presence of a 'flying a' is normal on early computers. The new character generator with descenders can be obtained as a 'word processing character generator' from Radio Shack.

### 87. Horizontal and vertical image adjustment can be done.

If the image is not centered on the screen (use

```
10 FORX=15360T016383:POKEX,191:NEXT
20 GOTO20
```

as a test program), adjust variable resistors R20 and R21 in the keyboard unit.

### 88. Adjust horizontal and vertical sync on the monitor.

If the image flickers badly, tears sideways, or rolls, the video monitor may be out of adjustment. Turn the white horizontal and vertical adjustment controls on the back of the monitor, just as with an ordinary television that exhibits the same symptoms.

### 89. Add a capacitor to cure video twitch.

A continuous, annoying screen twitch is the result of oscillations present at the video output. The insertion of a small capacitor (47 to 220 pF) between Z50 pin 3 and ground will eliminate the twitch.

### 90. Dimmers off! Get rid of the little runners on screen.

Similar to a twitch is the 'runner', a shaking horizontal line that works its way up or down through the screen, rocking one line of letters back and forth. This is a

kind of reverse RFI — not caused by the computer for a change — which can be cured by turning off light dimmers, faulty fluorescent or neon lights, or similar interference producers. If you live in an apartment building, neighboring dimmers should not (but may) affect your computer.

### 91. Adjust the monitor for blurred characters.

There is no high-voltage adjustment for blurred characters; however, there are a few kludges. First, a higher line voltage (a full 120 volts) will increase the sharpness. Adjusting the internal vertical height control may reduce the image to the more in-focus (center) area of the screen.

Also, replacing the power transistor on the bottom of the chassis and the high-voltage rectifier (they both may exhibit undesirable characteristics) can improve the image. Unfortunately, the monitor is a very basic video display, and has few adjustments.

### 92. Correct the lowercase software for LDOS.

The 'universal' (alas, a dangerous word) lowercase modification presented in this book is not universal at bootup for one disk operating system — LDOS. However, the LDOS driver can be invoked separately and will work; refer to the documentation.

### 93. Check the 5 volts or the optoisolator in the monitor.

A hardly existent or dim picture can be caused by insufficient 5 volts into the video cable from the computer (check for a bad connection), or a weak or dying optical isolator on the plug-in card inside the video monitor. First, try another monitor. If the system works with that monitor, resolder the cable connection if necessary; if that does not work, replace the optical isolator.

**XI. RFI**

*Symptoms of RFI (Radio Frequency Interference): herringbone across television screens, complete loss of TV stations, whistling on radio (AM or FM), disconnection of wireless telephones, blackout of shortwave transmissions.*

**94. Shield the entire system for RFI.**

The TRS-80 system is a broad-band interference generator; that is, what it sends out affects all bands of radio and television reception. The interference can be lowered by shielding: spray the inside of the case with aluminum paint and hook that to signal ground; use shielded multi-conductor cable for all peripherals (it's expensive); and create a 'Faraday cage' — if your decor will allow it — by shielding the room in which the computer is used with fine mesh.

**XII. HEAT**

*Symptoms of heat problems: loss or lockup of program consistently after the machine has been on several hours.*

**95. Ventilate case or add fan, latter if it's an all-in-one system.**

Normal ventilation using the slots cut into the TRS-80 and expansion box is adequate. However, if speed modifications, internal memory additions, etc., have been made, the power supply is asked to do extra work. A hard desk with good circulation around the computer is essential, and a small 'Sprite' type fan can be used occasionally to cool the system. A quiet fan may be added for continuous use, and is a necessity for an all-in-one-case system.

**96. Remove power supplies from the expansion interface.**

The two power supplies in the expansion interface generate a great deal of heat. With the video monitor stacked above,

this can create an undesirably hot environment for memory. Remove the two supplies from the expansion box; the system won't look as compact, but it will have a longer life.

**XIII. PRINTER**

*Symptoms of printer problems: will not backspace or underline, will not move up a line at a time during program listings.*

**97. Printer may need a line feed with every carriage return.**

Many printers require not only a carriage return, but a line feed as well; examples are Teletypes and some Centronics printers. If you have this problem, first check with the manufacturer for a modification kit or instructions. If none is available, use a disk operating system with a CR/LF option, a printer driver patch, or a text-editing system with the CR/LF option. A hardware addition can be made to most printer interfaces to generate a linefeed when a carriage return is received.

**XIV. USER PROCEDURE****98. Hide furry animals, keep away from wood stoves, etc.**

Insignificant as it may seem, smoke is a severe abrasive to disks. Smoking, wood stoves, unskilled kitchen use (ahem), animal hair, and so forth, can result in airborne particles that affect disks, whose rotation acts as a static vortex to pull in those particles. Similar cautions apply to the piles of hair and grit that can gather in the keyboard. If you wonder just how much dirt is in the air, open the video monitor (with the power off, of course), and have a look. All around the high voltage will be piles of grit pulled in because of the electrical attraction.

**99. Don't pull the cables while you're using the system.**

This might seem obvious, but realize that pressing the reset button jostles the cable, just as does moving the keyboard to make it comfortable. Dropping a pencil, a cassette, or a diskette case on the cable may generate noise; see Chapter 7 for vibration protection.

**100. Wait before power-up after power-down to protect memory.**

When the manual cautions to wait ten seconds before repowering the system, it is not merely because the system doesn't always fully reset during that time, but also because the application of power to the memory must be done in this order: -5 volts, 5 volts, 12 volts. The -5 volt line can be lost with too hasty repowering of the system, and memory will be physically damaged.

**101. Neither steel wool, nor metal filings, nor metal bits should be nearby.**

Using the computer in a home 'shop' is dangerous. Wood bits may cause key-bounce and disk damage, but metal filings may cause the entire system to fail. A buildup of metal dust will decrease local resistance levels to short-circuits, or create unexpected current drains and intermittent operation.

**102. No water or drinks nearby or open windows.**

Drinking cups can damage disks, water glasses can fall over, carbonated soda can bubble tiny sugar globs onto tapes and disks, and open windows can invite water-carrying breezes and even rainstorms.

**103. No magnets nearby, especially unobtrusive refrigerator types.**

Refrigerator or bulletin board magnets are handy things, but because they are so

ubiquitous, their danger to magnetic media is easy to forget. If there are any in the house, keep track of them . . . magnets shaped like daisies or fruits, bars, magnetic kiddie letters, even magnetic screwdrivers and scissors.

**104. Watch out for salt air areas.**

There's nothing like salt air for corroding metal, and metal is the heart of the computer's interconnections. Integrated circuit pins can corrode, cables can corrode, even screw connections can corrode. In boards previously reliable, salt air can corrode unseen bent pins, making the system flakey.

**105. Keep telephone bells away, which are bulk erasers.**

More than 50 volts shoot into a telephone bell's electromagnet. It's a strong magnetic field, and can act as a bulk eraser for disks sitting under them. Keep phones 'way back on the desk.

**106. Keep out of Xray at airports; hand check disks and tape.**

Xrays are damaging to magnetic materials. When going through an airport check, keep all your disks and tapes under your arm. Don't check them with your luggage, and don't let them go through the carry-on luggage conveyor belt Xray device. Insist on a hand check for those items.

**107. Attach cables correctly.**

Make sure all cables are correctly in place. Mark disk cables (the worst culprit) with 0, 1, 2 and 3, as well as 'top' and 'bottom'. Mark the expansion cable the same way (the metal wire or blue stripe is to the left, smooth side up). Mark all peripherals, particularly those which require power from the expansion box. Also, be sure the cassette cables are incorrectly (see #63 above).

**108. Attach cables with power off, no matter who says what.**

Occasionally, a manual may say something about turning the computer on, then attaching the cables. This is dead wrong. Correctly designed peripherals are always connected with the power off. If it is a construction project, avoid it; there's something wrong with the author's judgment. If it is a commercial product, return it; it shows poor design sense.

**109. Disconnect power especially during summer.**

High transient voltages can be present over the power and telephone lines during electrical storms. If your area is prone to electrical storms, keep your system unplugged — not just turned off. Also, disconnect direct-connect modems which can carry high voltages straight through to the rest of the system.

**110. Obtain a static-free mat for computer and peripherals.**

In dry climates, static buildup is common. A static zap can: change memory contents; damage memory; glitch a disk or cassette write; glitch a load; crash a program; reboot the system; or simulate just about any crash your system might be sensitive to. Obtain a static-free mat, or line your table with aluminum foil, and discharge to it — don't touch your computer first.

**111. Keep temperature 55 to 80 degrees, relative humidity 50 to 80 percent.**

This is simply good practice; although my computer is used in 40 degrees or below, disk reliability is lessened. Dry weather (low humidity) encourages static (see #110), and wet weather encourages corrosion (see #104).

**112. Test homebrew devices before installation.**

There is very little more to say; always

'proofread' your circuit with a second person, no matter how exhausting it may seem. It will prolong the life of your computer, and the homebrew device might even work the first time.

What are the chances that your problem will be one of these? Probably, the difficulty will be a combination. This list is derived from work on hundreds of TRS-80 system combinations and relatives. Virtually every suggestion has been made and every cure implemented. Sometimes the problems were multiple: one unit suffered from salt air, bent pins, corroded connectors, unbuffered cables, slow memory, and a bum program. Another was the victim of modifications made with a Boy Scout woodburning kit instead of a soldering iron.

Even if your problem is not included here, these suggestions should give you a clue about where and how to begin looking. Use the Radio Shack technical manual diagnostic chart — but don't believe the 'ROM is bad' section, because I've never seen a bad ROM. Instead, suspect a bus driver difficulty (Z22, 38, 39, 55, 75 and 76 in the keyboard). Otherwise, the manual will give you a good start on the tough stuff.

**Last Thoughts**

The solons at Radio Shack have done something very impressive: they have created a popular personal computer. From it have come the Models II and III, the Pocket Computers I and II, the Color Computers, the Model 16 is on the way, and a host of engineers across the world have been encouraged to come up with TRS-80 compatible hardware and software, and even full computer improvements like the LNW-80.

But they have done something unwittingly even more special: they have, through strict and strange corporate policies, challenge users to create the Custom TRS-80. Because computers become appliances more and more each day, there will only be one Custom TRS-80 . . . the humble Model I.

# NOTES

# INDEX

\* indicates discussion of topic  
# indicates a program listing  
\$ indicates a schematic diagram

## \*A\*

abbreviations \*17-18  
AC 97,163-164  
access, sequential 203  
access time 87,222,245  
accumulator 38  
accuracy, floating point 246  
A/D — see analog-to-digital  
address 23,24,40,42  
— absolute 135  
— bus 24,199-201  
addresses  
— 0050 lookup table 55  
— 0066 interrupt vector 27  
— 0075 re-entry point 26  
— 01C9 screen clear 28,\*36-39  
— 01FE tape drive on 79  
— 02B2 cassette entry 78  
— 03E3 keyboard scan \*53  
— 0674 initialization 25  
— 06CC Level II re-entry 27,37  
— 1987 syntax error 29  
— 1A19 Level II re-entry 29  
— 1BB3 user input 117  
— 1D5A interpreter entry 71-72  
— 1D78 interpreter loop 62  
— 1E5A integer conversion 78,117  
— 28A7 display routine 29,117  
— 3C00 video memory 38,61  
— 37E0-37EF mapped I/O 26,27,137,139,176  
— 3840 keyboard memory 26,53  
— 4000 patch points 25  
— 4012 interrupt vector 137,217  
— 4199 disk entry area 27  
— 42E9 start of program 28,117  
addressing  
— indexed 119  
— high-resolution graphics 143,148  
— peripheral 51  
AGC 96,98  
alpha decay 232  
ALPS keyboard 253  
Alternate Source, The 135  
ALU — see arithmetic logic unit  
AM radio 147  
analog 19,170  
analog-to-digital \*164-166  
animals, furry 231,255  
Apparat 112,205  
appliances 231,242  
arithmetic 24,57  
arithmetic logic unit 23  
architecture 23,24,53  
arrows 11  
ASCII 53-55,67,73,138,177

assembly programming 116

audio 156-158

autoexecution 71,#71

## \*B\*

B-17 loader 213  
Bach, Johann Sebastian 173  
bandwidth 96  
bank select \*189-193,\$190-191,#193  
Barden, William 119  
bases, numeric 32  
BASEX 154  
BASIC 19,36,37,40,57,62,65,71,115  
— programs 28  
— ROMs 22,24,38  
battery eliminators 13  
baud 203,212,213  
BCD 36  
beep 58,#59  
Beta-80 tape system 212  
bidirectional 136  
binary 21,24,\*32-34,35,36,79,178  
— digit 24  
— division 36  
bit 24,33,65,77,203  
— storage cost per 204  
black box 19,35  
Boolean algebra 34  
bootstrap 27,152  
break (function) 141  
break (key) 26,56,73,78  
bridges, solder 107  
buffer (hardware) 35,44,129,136,166,170,177,  
178-179,222  
— inverting 178  
buffer (software) 59,137,138,155  
buffered cable 15,91,122-124,244  
buffering 42  
bugs 242  
bus 23,44,135,187  
bus wire 10  
busy 177  
bypass capacitor 15,122,227  
byte 25,33,65,78,205  
BYTE 118,119

## \*C\*

cable  
— buffered 15,91,122-124,244  
— cassette 250  
— disk 248  
— interconnect 244,256  
— keyboard 21,22,\*229  
— multiconductor 17  
— rainbow 17  
capacitor 10,11,13,109  
— bypass 15,122,227  
— electrolytic 13  
— for video twitch 254  
— variable 11  
cartridge, eight-track 217,\*219-221  
cassette 42,58,130,132,203,211-212

- alignment 249
- cleaning 249
- diagnostic 238
- hardware description \*156-158
- input/output (software) \*78-83
- maintenance 235,249-250
- output circuitry \*48-49,\$49
- relay replacement 142,249,250
- reliability 203
- XRX modification 111,250
- central processing unit 22,23,24,27,42
- character generator 42,46,146,254
- chassis ground 11
- checksum 80,213
- CHR\$ 64,66,75,107,131
- cleaning \*86
- CLEAR 40
- CLOAD 67,71-72,182
- clock
  - flip-flop 35
  - CPU master \*42,\$42,134,136
  - high-resolution 146
  - high-resolution 146
  - real-time 121,\*182-188,\$183,\$185-186,  
#183-184,#187,#188-189,#192-193
  - speed increase 110-112,\$110,\$112
- clock-calendar 184-189
- cloning 161
- CLS \*36-39,#39
- CMD 137
- cold start 153
- color codes \*17
- Color Computer 41,143
- common ground 11
- comparator 222
- compiler 154
- composite video 96,104
- condition code 38
- conditions 70
- contact
  - keyboard
  - cleaner 99
- control codes 55
- conversions, numeric 32
  - hexadecimal 73
- converters — see digital-to-analog
- see analog-to-digital
- cosmetics \*96
- counter (hardware) 36
- counter (software) 54
- CPU — see central processing unit
- Craig tape deck 221
- crash 121,122,124,192,229,232
- crystal 11,146
- CSAVE 67,157,182
- CTR-41 tape deck 142,211
- CTR-80 tape deck 249
- custom interpreter 62,#62
- cutters 8

## \*D\*

- D/A — see digital-to-analog
- damage 85-86
- data 24
  - bus 24,199-201
- separator 246
- sheets following 36
- debouncing 58,#59
- decimal numbering 32
- decoding 35,42,110,121,167,170,177
- decrement 38,118
- demultiplexer 48
- derating, temperature 123
- despooler 138
- diagnostic routines 236-239
- Digi-Key Corporation 127,184
- digital delay line 126,251,252
- digital logic \*34-36
- digital-to-analog 164,171,173
- dimmers, light 254
- DIN connector 98,123
- diode 10,11
- DIP 45,214
  - shunt 90,252
  - switch 90,179
- dirt 248
- disk
  - alignment 247
  - BASIC 25
  - cleaning 247
  - controller 206-207
  - drive 19,26,27
  - floppy 204
  - maintenance 236,246
  - quality 246
  - reliability 203
- DOS 40,63,137,141,205,248
  - descriptions 209
- dots 46,104,143-151
- drills, hobby 94
- drive select signal 206
- Dust-Off 99
- Dvorak keyboard 76,94
- dynamic memory 44,45,77,87,200

## \*E\*

- earth ground 11
- edge card connector 12,23,43,121,122,131,136,  
\*175,\*187,235,243-244
  - cleaning \*233,247
  - plating 243-244
- edge connector (female) 127,187
- editing 58
- Editor/Assembler \*60-61
  - video driver modification #66-67
- EDTASM — see Editor/Assembler
- eight-track mass storage \*217-228,\$222-224,  
#225-228
- Electric Pencil 98,109,116,131,136
- electronic music — see music
- enable 104

Engineer's Notebook 10,13,164  
 ENTER key 28  
 envelope 170-174  
 EPROM 112,152  
   — bank select \*189-193,\$190-191,#193  
 error 58  
   — checking 155  
   — checksum 80,213  
   — L3 error 28,62  
   — OM error 29,64,90  
   — SN error 29,62,64,68,74  
   — soft 232  
   — tape 203-204  
 Exatron 233  
 Exatron Stringy-Floppy 83,123,167,204,\*210-211,213,217  
   — maintenance 236  
 exclusive OR 179  
 execution routine 155  
   — auto 71,#71  
 expansion interface 20,121  
   — bus 187  
   — clock speed-up 126  
   — memory expansion \*91  
   — opening \*124-126  
   — reset modification \*91-92,252  
 extension cords 231

**\*F\***

Farad 17  
 Fastload tape system 203,\*211-212  
 FDC — see floppy disk controller  
 file (metal) 8,244  
 filter, low-pass 49  
 filtering 14,164,242  
 flip-flop 35,36,183  
 floating point accuracy 246  
 floppy disk controller 26,27,205,\*206-207  
 fluorescent lights 231  
 flux — see solder, flux  
 formatting, eight-track 225  
 form-feed 70  
 FOR-NEXT 38,40  
 FORTH 154  
 FORTRAN 115,154  
 front panel \*199-201,\$201  
 functional schematics 12  
 furry animals 231,255  
 fuse 242

**\*G\***

garbage collection 241  
 gates \*34-36  
   — AND 34  
   — Exclusive OR 179  
   — NAND 35,141,142  
   — NOR 35  
   — OR 34,38,179  
   — TTL diagrams following 36  
 gold plating 243  
 GOSUB 36,37,40

GOTO 36,37  
 graphics characters 65  
 graphics, high-resolution \*143-151,\$144-146,\$147,#149  
 Gray code 36  
 grommets 21  
 ground 11,\*89,123,242  
 Gunn, David 167

**\*H\***

HALT diagnostic 237  
 hardware  
   — description 42  
 head, read/write 206  
 heartbeat 25  
 hexadecimal 18,66  
 high-level language 115  
 high-speed  
   — diagnostic 238  
   — ladders 213  
   — see also clock, speed increase  
 Hobbyworld Electronics 184  
 Hofstadter, Douglas 65  
 home-cursor 70  
 hook 63  
 hot chassis 97  
 house current 14,19  
 humidifier 231

**\*I\***

I/O — see input/output  
 increment 38,80,118  
 index  
   — hole 205  
   — pulse 206  
 indexed addressing 119  
 initialization 137,225  
   — see also power-up  
 INKEY\$ 27,47,53,58,59,75-76  
 input 19,162  
   — diagnostic 237  
 INP 177  
 INPUT 28,78  
 input/output 20,24  
 instruction 23,167  
   — AND 26,53  
   — CALL 26,37,78-79,135  
   — DI 25,27  
   — DJNZ 26  
   — EI 137  
   — HALT 91,141  
   — IM0, IM1, IM2 135-136  
   — JUMP 37  
   — LDIR 25,27  
   — NOP 141  
   — OR 28  
   — RLC 53  
   — RRA 81  
   — RST 28,56,135  
   — XOR 25,53,80  
 integrated circuit 12,13,19  
   — families 13

- handling \*89
- pins 33
- interfacing 162-164,\$163,170,\*179-181,\$180
- printer \*175-177
- synthesizer \*166-174
- interpreter 24,58,\*62-65,78,\*154-155
- custom #62
- interrupt 25,27,\*134-140,183,217
- diagnostic
- modes 136
- inverter 35

#### \*J\*

Jameco Electronics 98-99  
 jitter 147  
 jumpers, programming 22

#### \*K\*

KBFIX 99,117,253  
 keyboard 58,130,132

- additional \*93-96
- cable 21,22,88,\*229-230
- cleaning 99,253
- comparison (photos) 105
- debouncing #59,99
- display routine #55
- general description 24,42
- hardware description \*46-47,\$46
- hexadecimal \*98-102,\$101
- lockout 86
- matrix 38
- PEEK process 75
- scan \*53-56

keyboard (CPU) unit 143

- memory expansion 89-90

keybounce 131

- see also debouncing, keyboard

keycaps 99,235

- lifting tool 99

#### \*L\*

Lab Service Inc. 142  
 label 61  
 Lancaster, Don 98  
 latch 36,48,121,166-167,\$166,199-201,\$201  
 leap year 184  
 LED 87-88,100,162,199

- infrared 214
- power 230
- tri-color 112

Level I 22,38,\*40-41,42,112

- combined with Level II \*112-114
- ROMs 47,89,152

Level II 22,24,28,37,38,\*40-41,42,57,58,63,66,136,139,189

- combined with Level I \*112-114
- in PMC-80 129
- interconnect cable 91,98
- ROMs 48,\$48,68,98,111,152
- ROMs, two-chip 111,113

Level III 25,\*40-41,62,182,209

Lien, David 112  
 lightning 232

- arrestor 243

Lindsly, Jerry 120  
 line numbers 115  
 linear predictive coding 169  
 lines, high-resolution 148,#149-150  
 LISP 154  
 LIST 68,\*70  
 LLIST 70,138,175  
 LNW 15,92,126

- expansion board \*126-129,252
- LNW-80 132

loaders, special 81-83  
 lowercase 56,58-60,#59-60,\*106,\$106,254

- see also uppercase, video memory

low-level language 115  
 LPRINT 64,65,138,175  
 LSI 89,121  
 lubrication 247

#### \*M\*

machine cycle 141,167  
 machine language \*36-39,40,66-69,135,212

- instruction 167 see also instruction
- monitor 72-73,#72-74,153,#154-158

magnets 256  
 maintenance of system 235-236  
 mass storage \*203-228,\$216,\$222-224,#213-216,#217-219,#225-228  
 mathematic functions 58  
 measurement 161  
 memory 24

- bank select \*189-193,\$190-191,#193
- conservation of 116
- crashes 121,244-245
- dynamic 44,45,77,87,122
- expansion \*86-91
- map 24,\*38-39,189,192
- map, high-resolution 148
- map table 39
- mapping 26,184
- older 245
- slow 244
- write-protected 192
- speed (access time) 87,222
- test 77,#74-76

memory sidecar 192,\$194-195,196,#197-199  
 MEMORY SIZE? 25,28,29,36,61,64,81,90,113,124,142,153,233,241

- resetting \*74
- setting 29,\*40

microphonics 91,122  
 microprocessor 23,152,199  
 Microsoft 114  
 Microtek 15  
 mode (interrupts) 135-136  
 monitor (hardware) \*199-201,\$201  
 monitor (software)— see machine language monitor  
 monitor, power supply 242  
 monitor, video 122

- see also video monitor

motor 162,205  
— stepping 204  
multiconductor cable 10  
multimeter 9  
multiplexer 36  
multiplexing 42  
music \*166-174,\$166-167,\$170-\$172,#169,  
#171-174

#### **\*N\***

nanofarads 15  
nanoseconds 87  
National Parts 23  
nesting 155  
NEW 58,\*73-74,#74  
NMI \*135,141  
noise 15,91,122,123,129,233  
non-maskable interrupt 135  
number systems \*31-34  
numeric conversions 57  
nut driver 8

#### **\*O\***

ohms 13,17  
OPEN 73  
open-collector 162,\$163,172  
opening the computer 21,\*87-88  
opening the expansion interface \*124-126  
operand 61,119  
operational amplifier 222  
opto-isolator 254  
oscilloscope 9,157,\*234-235  
OUT 103-105,111-112,114,173,177,192  
output controls \*48-50,162  
— diagnostic 237

#### **\*P\***

package 24,33  
packing 66  
pad 12  
paint 96  
PAIA 169  
paper tape reader \*214-217,\$216,#217-219  
parentheses 38  
patch point 63,116  
PEEK 65,75-76,177,232,241  
peripheral 169  
— interface, programmable 171,179  
Personal Micro Computers Inc. 129  
phantom bit 107  
phase-shift 50  
piggyback  
— integrated circuits 103,106-108,111,113  
— sockets 93  
pin numbers 12  
— diagrams, TTL following 36  
— how to read \*33  
pins, bent 245  
pixel, 143  
pliers 8  
PMC-80 \*129-134  
POKE 65,66,108,149,177,232,241

polarity 13  
polling 136  
Poor Man's Floppy — see TC-8  
port 170,179,192  
— 'port FE' 49  
— 'port FF' 25,49,59,78-80,177  
power 11  
— cables 242  
— drains 242  
— strip 231  
— see also power supply; AC; house current;  
voltage  
power supply 12,\*13-16,\$13-16,22,42,50,89,121,  
132,147  
— clock \$186  
— high-resolution graphics \$145  
— problems 233  
— synthesizer \$171  
— transistor 251  
power-up 256  
— monitor 153  
— routines \*25-29,57  
pressure pad 205-206  
PRINT 64,65,68,108  
printer 19,58,64,121,139,\*175-177,\$176  
— diagnostic 238  
— maintenance 235,255  
probe, oscilloscope 234  
program counter 118,135  
pullup, pulldown — see termination

#### **\*Q\***

Qwerty 76,98

#### **\*R\***

Raeco 214  
rainbow cable 15  
RAM 38,42  
— damaged 86  
— dynamic 44,87,232  
— expansion 86-89  
— reserved 38,53  
— static 192,232,257  
— using 4K for expansion 127  
— video 45,106  
— write-protected 192  
— see also memory, dynamic memory, ROM  
READ 69,246  
read circuitry diagnostic 237  
read-only-RAM — see RAM, write-protected  
read/write 42,205  
READY 27,37,72,91,113,141,211  
recording 156  
reference manuals 23  
refresh 42,44,86,122,152,200  
registers 23,37,119  
— control 188  
regulation 12,\*13-16  
regulators 12  
relative branch 118  
relative subroutine 119  
relay, cassette 142,249

- reliability, tape 78-83,203
- relocatable code \*115-120
- REM 65,70
- repeating keys 58,#59
- reset button 92,135,141
  - expansion modification \*91-92
  - keyboard modification \*92-93,252
- resistor 10,11,12,13
  - termination 15,92,123,128,251,252
  - variable 11
- restart 114
- reverse video \*103-105,\$103
- RFI (radio frequency interference) 255
- RF modulator 50,96,\*98,130,132
- rollover 54
- ROM 38,40,42,60,68,98,116,120,131,138
  - bank select \*189-193,\$190-191,#193
  - control \*47-48
  - Level I \*112-113
  - replacement \*152,\$152
- RPGII 115
- RS-230 121-123,246
  - maintenance 235,246
- RUN 71
- \*S\*
- salt water 231,256
- Sams publications 10
- scalpel 8
- schematics \*10-12
- Schmitt trigger 162
- scratches 96
- screwdriver 8
- Scriptsit 131
- scroll 136-137
- sector 205
- sensor, foil 222
- serial-to-parallel conversion 79,157
- serialization 156
- servicing interrupts \*135-140
- shield 14
  - RFI 255
- shift key 54-55,95
- shift register 36
- shock hazard 97
- shockproofing (vibration) 123,232
- shunt 45,\*90,252
- signal 34,200
- signalling 161
- signals
  - BLANK 46
  - CAS 44,45,122,126,153,245
  - DAL0-DAL7 207
  - HALT 135
  - HDRV 50
  - IN 43,153,222
  - INSIG 48,49
  - INT 47,135,182,215
  - INTAK 215
  - INTRQ 207
  - IORQ 43
  - KYBD 46,47
  - MEM 47
  - MREQ 43,111,122
  - MUX 44,45,122,126,153,245
  - OUT 43,103,153,222
  - OUTSIG 48,49,78
  - RAS 45,122,126,141,153,245
  - RD 43,111,153,177
  - RESET 43
  - RFSH 44
  - SYSRES 114
  - TEST 44,153
  - VDRV 50
  - VID 45,47
  - WR 43,45,153,177,192
- Silver-It 86,243
- slash 80
- smearing, video 98
- smoke 231
- socket 89,121,124
- soft error 232
- solder 9
  - braid 9
  - bridges 107,252
  - flux 52,86,107
  - iron 8,52,89
  - silver 86
  - technique \*52
  - wick 9,106
- solder joints, cold 252
- soldering iron grounding 89
- sound #67-69,\*166-174,\$166-167,\$170-172
  - effects 70,#70,\*169-173
- spacers — see grommets
- speed increase, CPU — see clock, speed increase
- speed of execution 116-118
- spikes 14
- splice, tape 219
- spooler 138,#139
- stack 40,138,241
- static 89,184
- static-free workbench 89,257
- start bit 79
- stepping motor — see motor, stepping
- storage 157
- string 40,58
  - packing \*66-69
- Stringy-Floppy — see Exatron Stringy-Floppy
- subroutines 116,118
- switch box, cassette 250
- symbol table 61
- symbols \*10-12
- sync (hardware) 42,50,96
- sync (software) 79
- synchronization
  - oscilloscope 234
  - power line 182
  - process 79
- syntax 155
- synthesizer \*169-174
- SYSTEM 71,\*78-80,116

## **\*T\***

tape, cassette — see cassette  
tape, paper — see paper tape reader  
TC-8 tape system \*211-212  
Technical Reference Handbook 12,23,42,44-47,50,  
78,106  
telephone  
— bells 256  
— communications 19  
Teletype 139  
television 50,96,\*98,130,132,231  
temperature derating 123  
termination resistor 15,123,128,251,252  
text editing 136  
Thermo-Fax belt cleaner 107  
TIME\$ 182-183  
tinning 230  
token 62,\*63-65,154  
tolerances \*12-13  
tone color 169  
tools \*8-10  
TPR-1 paper tape reader \*214-217  
traces 19  
transformer 11  
transistor 10,11  
transitions 70  
trigger (envelope) 171  
trim erase 204  
trimpot 107  
TRS-80  
— Color Computer 41  
— general description \*19-21  
truth table 34  
TTL 122,162  
turnkey system 23  
tweezers 8  
twisted pair 122,245

## **\*U\***

UART 129  
uppercase 56,58-59,#59-60  
— see also lowercase, video memory  
USR 68

## **\*V\***

variable (resistor, capacitor) 11  
variables 40,58,155,241  
— string 66  
VARPTR 66-69,241  
VCO 172  
vector 63,116  
vibration 123,232  
video 25,38,58,130,136  
— alignment 235,254  
— buffer 253  
— composite 96,104  
— countdown chain 146  
— description 24,42  
— diagnostic 237  
— driver routine #63-66  
— in PMC-80 132  
— monitor, description 103

— monitor, additional \*96-98,\$97  
— output circuitry \*50,\$50  
— RAM (memory) 45,46,65,106,143  
— reverse (complete screen) \*103-105,\$103  
— reverse (individual characters) \*106-110  
— scan (beam) 143  
— tear 253  
— twitch 253  
voice input/output 19,#81-82,\$83  
volt 17,162  
voltage  
— control 172  
— isolation 162  
— levels 24,164  
— power supply 12  
Votrax 169

## **\*W\***

wafer 204  
warm start 153  
Watt 17  
waveform 80,169  
wire, bus 10  
wire-wrap  
— socket 93  
— technique \*51  
— tool 8,51  
— wire 10,51  
write circuitry diagnostic 236  
write enable 206  
write-protect (disk) 204,206  
write-protect (memory) 192

## **\*X\***

X-acto knife 8,96,106,107  
X-ray danger 256  
XRX modification 111,250  
X3 (shunt) — see Z3  
X71 (shunt) — see Z71

## **\*Y\***

## **\*Z\***

Z3 22,47,90,113,252  
Z71 22,45,90,252  
Z-80 22,37,53,60,89,117,141,167  
— block diagram 23  
— handling 89  
— interrupts \*134-140  
— signals 43  
— technical manual 135  
Z80A 251

# NOTES

---

## Appendix I

### Parts Suppliers

#### The Four Stars

**Digi-Key Corporation**, Hiway 32 South, P.O. Box 677, Thief River Falls, MN 56701. 800 346-5144. *COD, check, money order, credit cards. Volume discounts over \$100; shipping, insurance prepaid.*

This company is in my opinion the hobbyist's best. Shipping is fast (five days from ordering to my door in Vermont), and most items are in stock. Their catalog is monthly, and items not stocked are not listed. All merchandise is prime; no bubble packs.

**QT Computer Systems**, 15335 S. Hawthorne Blvd., Lawndale, CA 90260. 800 421-5150. *COD under \$100, check, money order; credit cards preferred. Quantity discounts, no insurance.*

A good hobbyist catalog similar to Digi-Key, with competitive prices. This is a new company, but they are already beginning to make a mark for promptness, exceeding courtesy, and prime parts. Their catalog is very complete and quite up-to-date.

**Advanced Computer Products**, 1310 E. Edinger, Santa Ana, CA 92705. 800 854-8230. *COD, check, money order, credit cards. Volume discounts, no insurance.*

One of the best catalogs in the business, prompt, but be wary of substitutions in orders. Specify voltages of devices and check upon receipt. Expect harassment from Customer Service. Otherwise, they have what you can't get anywhere else.

**Jameco Electronics**, 1355 Shoreway Road, Belmont, CA 94002. 415 592-8097. *COD, check, money order, no credit cards. No discounts, no insurance.*

One almost wonders why to put Jameco in with the four best, but their selection is contemporary and their response prompt. They have items others don't have in stock for the popular computer hobbyist. Bubble pack stuff on retail store racks at Lafayette Radio and others. Highest prices in the business.

#### And Others

**Jade Computer Products**, 4901 West Rosecrans Ave., Hawthorne, CA 90250. 800 421-5500. *No CODs; checks, money order; credit cards preferred. Quantity discounts; insurance under 50 lbs.*

This company works hard at immediate hobbyist needs and some unusual items. Get their catalog, but consult monthly ads in electronics magazines for hot items.

**Priority One Electronics**, 16723C Roscoe Blvd., Sepulveda, CA 91343. 800 423-5633. *No CODs; check, money order, credit cards. Quantity discounts, insurance.*

Priority deals for the most part in larger items for computer hobbyists, with only a token selection of small parts. This company concentrates on boards and naked disk drives, and heavier hardware.

**Hobbyworld Electronics**, 19511 Business Center Dr., Northridge, CA 91324. 800 423-5387, (800 382-3651 in CA). *COD (\$1.25 extra), check, credit cards. No discounts, no insurance.*

Hobbyworld is the computer hobbyist's pop culture. It stocks all the hot items with a high turnover. Look to them for low prices on items you need right away.

**Electrolabs**, P.O. Box 6721, Stanford, CA 94305.

The best part is always their funny and schizophrenic catalog with an honest selection and a wealth of good information.

For example . . .

"Save yourselves \$6.75 and use a 25 cent transistor the next time your looking for a temperature probe." Also, TTL Family rules of Incest are great.) The shipping was always prompt and the merchandise prime.

#### Not Recommended

**Active Electronic Sales**, P.O. Box 1035, Framingham, MA 01701. 617 879-0077. *Minimum \$10, handling \$2, check (wait to clear), money order. No discounts, no insurance.*

This group claims to be "The World's Largest International Semiconductor Distributor", which implies lots of stock, in stock. No way. All my orders have been returned 25 percent filled, with 50 percent errors.

---

## Appendix II. Bibliography.

## Reference Books

### Manuals, Guides, and Data Books

Radio Shack, Tandy Corporation. Fort Worth, Texas.

*TRSDOS & Disk BASIC Reference Manual*, 1979.

*TRS-80 Micro Computer Technical Reference Handbook*, 1978.

Printer Interface Cable, Service Manual.

Expansion Interface, Service Manual.

TRS-80 16K RAM Expansion, Service Manual. With Addendum.

National Semiconductor, 2900 Semiconductor Drive, Santa Clara, California 95051.

*TTL Databook*, 1975.

*Memory Databook*, 1977.

*CMOS Databook*, 1977.

*Series 8000 Microprocessor Family Handbook*, 1978.

— *Special Functions Databook*, 1979.

— *Linear Data Book*, 1976.

— *Linear Applications*, Volume 1, 1973.

— *Linear Applications*, Volume 2, 1977.

— *Voltage Regulator Handbook*, 1975.

— *Pressure & Temperature Transducers*, 1974.

Motorola, Inc. *The Complete Motorola Microcomputer Data Library*. Technical Information Center, Box 20912, Phoenix, Arizona 85036. 1978.

Pro-Log Corporation. *Microprocessor User's Guide*. 2411 Garden Road, Monterey, California 93940. 1980.

Zilog, Inc. *Z80-CPU / Z80A-CPU Technical Manual*. 10340 Bubb Road, Cupertino, California 95014. 1977.

Jonathan A. Titus, Christopher A. Titus, and David G. Larsen, *TRS-80 Interfacing. Books 1 and 2*. Each 250 pp. Howard W. Sams and Co., Inc., 4300 West 62nd St., Indianapolis, Indiana 46268. 1979, 1980.

Sybex, 2020 Milvia Street, Berkeley, California 94704.

Rodnay Zaks, *Microprocessors, from Chips to Systems*. 416 pp. 2nd ed., 1977.

Rodnay Zaks, *Programming the Z80*. 624 pp. 1979.

Austin Lesea and Rodnay Zaks, *Microprocessor Interfacing Techniques*. 416 pp. 2nd ed., 1978.

William Barden, Jr., *The Z-80 Microcomputer Handbook*. 304 pp. Howard W. Sams and Co., Inc., 4300 West 62nd St., Indianapolis, Indiana 46268. 1978.

William Barden, Jr., *TRS-80 Assembly Language Programming*. 224 pp. Radio Shack, Fort Worth, Texas. 1979.

Adam Osborne, Jerry Kane, Russell Rector, and Susanna Jacobson, *Z80 Programming for Logic Design*. Osborne and Associates, P. O. Box 2036, Berkeley, California 94702. 1978.

Jim Perry and Chris Brown, eds., *80 Programs for the TRS-80*. 234 pp. 1001001 Inc., Peterborough, New Hampshire 03458. 1979.

John Blattner and Bryan Mumford, *Inside Level II, A Programmer's Guide to the TRS-80 ROMs*. 65 pp. Mumford Micro Systems, Box 435, Summerland, California 93067. 1980.

H. C. Pennington, *TRS-80 Disk and Other Mysteries*. IJG Computer Services, 1260 W. Foothill Blvd., Upland, California 91768. 1979, 1980, 1981.

James Farvour, *Microsoft BASIC Decoded and Other Mysteries*. IJG Computer Services, 1260 W. Foothill Blvd., Upland, California 91768. 1981.

David A. Lien, *User's manual for Level 1*. 232 pp. Radio Shack, Fort Worth. 1978.

W. J. Weller, *Practical Microcomputer Programming: The Z80*. 480 pp. Northern Technology Books, Box 62, Evanston, Illinois 60204. 1978.

Thomas C. McIntire, *Software Interpreters for Microcomputers*. 233 pp. John Wiley and Sons, New York. 1978.

Robert Richardson, *Disassembled Handbook for TRS-80*. (In continuing volumes; 1, 2 and 3 published to date). Richcraft Engineering Ltd., Drawer 1065, Chautauqua, New York 14722. 1980.

Forest M. Mims, III, *Engineer's Notebook: A Handbook of Integrated Circuit Applications*. 128 pp. Radio Shack, Fort Worth, 1979.

Paul Warne, *BASEX: A Simple Language and Compiler for 8080 Systems*. With TRS-80 addendum. 97 pp. BYTE Books, 70 Main Street, Peterborough, New Hampshire 03458. 1979.

Forth, Inc., *Microforth Primer*. FORTH, Inc., 815 Manhattan Avenue, Manhattan Beach, California 90266. 1978.

Hal Chamberlin, *Musical Applications of Microprocessors*. 660 pp. Hayden Book Company, Inc., Rochelle Park, New Jersey. 1980.

Wayne Bateman, *Introduction to Computer Music*. 314 pp. John Wiley and Sons, New York. 1980.

G. Michael Schneider, Steven W. Weingart, and David M. Perlman, *An Introduction to Programming and Problem Solving with Pascal*. 394 pp. John Wiley and Sons, New York. 1978.

### Periodicals and Irregulars

*80 Microcomputing*. 80 Pine St., Peterborough, New Hampshire 03458. \$18 per year.

*Kilobaud Microcomputing*. 73 Pine St., Peterborough, New Hampshire 03458. \$25 per year.

*BYTE*. 70 Main St., Peterborough, New Hampshire 03458. \$19 per year.

*onComputing*. 70 Main St., Peterborough, New Hampshire 03458. \$12 per year.

*80-U.S. Journal*. 3838 South Warner Street, Tacoma, Washington 98409. \$24 per year.

*Softside*. 6 South Street, Milford, New Hampshire 03055. \$24 per year.

*Popular Electronics*. One Park Avenue, New York, New York 10016. \$14 per year.

*The Alternate Source*. 1806 Ada Street, Lansing, Michigan 48910. \$12 per year.

*Radio Electronics*. 200 Park Avenue South, New York, New York 10003. \$9.98 per year.

*CLOAD*. P.O. Box 1267, Goleta, California 93017. \$42 per year, on cassette.

Fairfield County Users Group, *Voice of the 80*. C/O Alan Abrahamson, 10 Richlee Road, Norwalk, Connecticut 06851.

*Marin County TRS-80 Users Group Newsletter*. P.O. Box 895, Novato, California 94948.

*Computer Base Lubbock*. C/O Roger Smith, 2601 Nonesuch Dr., Lot 1802, Abilene, Texas 79606.

*TCS*, Club Project of the Tidewater TRS-80 Users Group. P.O. Box 10281, Norfolk, Virginia 23513.

*Northern Bytes*. Microcomputer Users International. C/O Jack Decker, 1804 West 18th Street, Lot 155 Sault Suite. Marie, Michigan 49783.

*Orange Country TRS-80 Users Group Newsletter*. C/O Ed Faulk, 2531 E. Commonwealth, Fullerton, CA 92631.

*The Bit Bucket*. Texhoma Microcomputer Enthusiasts, P.O. Box 1384, Wichita Falls, Texas 76301.

*Amateur Computer Group of New Jersey News*. UCTI, 1776 Raritan Road, Scotch Plains, New Jersey 07076.

*Delaware Valley Computer Club Newsletter*. P.O. Box 651, Levittown, Pennsylvania 19058.

Cleveland Digital Group, *TRS-80 User's Group Newsletter*. C/O Cleveland Heights Public Library, 2345 Lee Road, Cleveland Heights, Ohio 44112. Or C/O 1838 Willowhurst Ave., Cleveland, Ohio 44112.

*Chicatrug News*. Chicago TRS-80 Users Group. Emmanuel B. Garcia Jr., and Associates, 203 N. Wabash, Room 2102, Chicago, Illinois 60601.

### Thought and Relaxation

Frederick P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1978.

Douglas Hofstadter, *Godel, Escher, Bach: The Eternal Golden Braid*. Vintage Books. 1980.

# NOTES

APPENDIX 3  
Byte Values and Their Equivalents

Code		POKE displays:			PRINT:	Edtesm	Standrd
Hax Val	Oac Val	Without L/Case	With Old LC	With New LC	w/ LC Driver	Z80 Opcode	ASCII Meaning
00	0	@	gaming	@	none	NOP	NUL
01	1	A	gaming	A	nona	LO BC,NN	SOH
02	2	B	gaming	B	none	LO (BC),A	STX
03	3	C	gaming	C	none	INC BC	ETX
04	4	O	gaming	O	nona	INC B	EOT
05	5	E	gaming	E	nona	DEC B	ENQ
06	6	F	gaming	F	nona	LO B,N	ACK
07	7	G	gaming	G	none	RLCA	BEL
08	8	H	gaming	H	bksPCA	EX AF,AF'	BS
09	9	I	gaming	I	bksP lin	ADD HL,BC	HT
0A	10	J	gaming	J	linafaad	LO A,(BC)	LF
0B	11	K	gaming	K	nona	OEC BC	VT
0C	12	L	gaming	L	formfeed	INC C	FF
0D	13	M	gaming	M	car retn	DEC C	CR
0E	14	N	gaming	N	crsr on	LD C,N	SO
0F	15	O	gaming	O	crsr off	RRCA	SI
10	16	P	gaming	P	nona	OJNZ d	OLE
11	17	Q	gaming	Q	none	LO DE,NN	DC1
12	18	R	gaming	R	none	LO (OE),A	OC2
13	19	S	gaming	S	none	INC OE	OC3
14	20	T	gaming	T	none	INC O	OC4
15	21	U	gaming	U	nona	DEC O	NAK
16	22	V	gaming	V	none	LO O,N	SYN
17	23	W	gaming	W	widemode	RLA	ETB
18	24	X	gaming	X	bksPCrsr	JR d	CAN
19	25	Y	gaming	Y	adv crsr	ADD HL,DE	EM
1A	26	Z	gaming	Z	dn linfd	LO A,(DE)	SUB
1B	27	l brace	gaming	up arr	up linfd	DEC DE	ESC
1C	28	dn arr	gaming	dn arr	home crsr	INC E	FS
1D	29	r brace	gaming	leftarr	strt crsr	DEC E	GS
1E	30	rt arr	gaming	rt arr	arasa lin	LO E,N	RS
1F	31		gaming		clr frama	RRA	US
20	32	space	space	spaca	space	JR NZ,d	space
21	33	!	!	!	!	LO HL,NN	!
22	34	"	"	"	"	LO (NN),HL	"
23	35	#	#	#	#	INC HL	#
24	36	\$	\$	\$	\$	INC H	\$
25	37	%	%	%	%	OEC H	%
26	38	&	&	&	&	LO H,N	&
27	39	'	'	'	'	DAA	'
28	40	(	(	(	(	JR Z,d	(
29	41	)	)	)	)	ADD HL,HL	)
2A	42	*	*	*	*	LD HL,(NN)	*
2B	43	+	+	+	+	DEC HL	+
2C	44	,	,	,	,	INC L	,
2D	45	-	-	-	-	DEC L	-
2E	46	.	.	.	.	LO L,N	.
2F	47	/	/	/	/	CPL	/
30	48	0	0	0	0	JR NC,d	0

Code		POKE displays:			PRINT:	Edtasm	Standrd
Hex Val	Dec Val	Without L/Case	With Old LC	With New LC	w/ LC Driver	Z80 Opcode	ASCII Meaning
31	49	1	1	1	1	LD SP,NN	1
32	50	2	2	2	2	LD (NN),A	2
33	51	3	3	3	3	INC SP	3
34	52	4	4	4	4	INC (HL)	4
35	53	5	5	5	5	DEC (HL)	5
36	54	6	6	6	6	LD (HL),N	6
37	55	7	7	7	7	SCF	7
38	56	8	8	8	8	JR C,d	8
39	57	9	9	9	9	ADD HL,SP	9
3A	58	:	:	:	:	LD A,(NN)	:
3B	59	;	;	;	;	DEC SP	;
3C	60	<	<	<	<	INC A	<
3D	61	=	=	=	=	DEC A	=
3E	62	>	>	>	>	LD A,N	>
3F	63	?	?	?	?	CCF	?
40	64	@	open quote	@	@	LD B,B	@
41	65	A	A	A	A	LD B,C	A
42	66	B	B	B	B	LD B,D	B
43	67	C	C	C	C	LD B,E	C
44	68	D	D	D	D	LD B,H	D
45	69	E	E	E	E	LD B,L	E
46	70	F	F	F	F	LD B,(HL)	F
47	71	G	G	G	G	LD B,A	G
48	72	H	H	H	H	LD C,B	H
49	73	I	I	I	I	LD C,C	I
4A	74	J	J	J	J	LD C,D	J
4B	75	K	K	K	K	LD C,E	K
4C	76	L	L	L	L	LD C,H	L
4D	77	M	M	M	M	LD C,L	M
4E	78	N	N	N	N	LD C,(HL)	N
4F	79	O	O	O	O	LD C,A	O
50	80	P	P	P	P	LD D,B	P
51	81	Q	Q	Q	Q	LD D,C	Q
52	82	R	R	R	R	LD D,D	R
53	83	S	S	S	S	LD D,E	S
54	84	T	T	T	T	LD D,H	T
55	85	U	U	U	U	LD D,L	U
56	86	V	V	V	V	LD D,(HL)	V
57	87	W	W	W	W	LD D,A	W
58	88	X	X	X	X	LD E,B	X
59	89	Y	Y	Y	Y	LD E,C	Y
5A	90	Z	Z	Z	Z	LD E,D	Z
5B	91	l.brkt. l.brkt. up arr.			up arr.	LD E,E	l.brkt.
5C	92					LD E,H	slant
5D	93					LD E,L	r.brkt.
5E	94					LD E,(HL)	carat
5F	95					LD E,A	l.arr.
60	96	@	open quote	pound	@	LD H,B	undef.
61	97	A	a	a	a	LD H,C	a
62	98	B	b	b	b	LD H,D	b
63	99	C	c	c	c	LD H,E	c
64	100	D	d	d	d	LD H,H	d
65	101	E	e	e	e	LD H,L	a

Code		PDKE displays:			PRINT:	Edtasm	Standrd
Hex Val	Dec Val	Without L/Case	With Old LC	With New LC	w/ LC Driver	Z8D Dpcode	ASCII Meaning
66	102	F	f	f	f	LD H,(HL)	f
67	103	G	g	g	g	LD H,A	g
68	104	H	h	h	h	LD L,8	h
69	105	I	i	i	i	LD L,C	i
6A	106	J	j	j	j	LD L,D	j
6B	107	K	k	k	k	LD L,E	k
6C	108	L	l	l	l	LD L,H	l
6D	109	M	m	m	m	LD L,L	m
6E	110	N	n	n	n	LD L,(HL)	n
6F	111	O	o	o	o	LD L,A	o
70	112	P	p	p	p	LD (HL),8	p
71	113	Q	q	q	q	LD (HL),C	q
72	114	R	r	r	r	LD (HL),D	r
73	115	S	s	s	s	LD (HL),E	s
74	116	T	t	t	t	LD (HL),H	t
75	117	U	u	u	u	LD (HL),L	u
76	118	V	v	v	v	HALT	v
77	119	W	w	w	w	LD (HL),A	w
78	120	X	x	x	x	LD A,8	x
79	121	Y	y	y	y	LD A,C	y
7A	122	Z	z	z	z	LD A,D	z
7B	123					LD A,E	l.brace
7C	124					LD A,H	separator
7D	125					LD A,L	r.brace
7E	126					LD A,(HL)	wave
7F	127					LD A,A	delete

#### Upper Character Set

HEX Val	DEC Val	PDKE Display:	BASIC Keyword:	PRINT Display:	Z8D Dpcode:
80	128	/G/	END	/G/	ADD A,8
81	129	/G/	FOR	/G/	ADD A,C
82	130	/G/	RESET	/G/	ADD A,D
83	131	/G/	SET	/G/	ADD A,E
84	132	/G/	CLS	/G/	ADD A,H
85	133	/G/	CMD	/G/	ADD A,L
86	134	/G/	RANDDM	/G/	ADD A,(HL)
87	135	/G/	NEXT	/G/	ADD A,A
88	136	/G/	DATA	/G/	ADC A,8
89	137	/G/	INPUT	/G/	ADC A,C
8A	138	/G/	DIM	/G/	ADC A,D
8B	139	/G/	READ	/G/	ADC A,E
8C	140	/G/	LSET	/G/	ADC A,H
8D	141	/G/	GOTO	/G/	ADC A,L
8E	142	/G/	RUN	/G/	ADC A,(HL)
8F	143	/G/	IF	/G/	ADC A,A
90	144	/G/	RESTORE	/G/	SUB B
91	145	/G/	GOSUB	/G/	SUB C
92	146	/G/	RETURN	/G/	SUB D
93	147	/G/	REM	/G/	SUB E

---

Upper Character Set

---

HEX Val	OEC Val	POKE Display:	BASIC Keyword:	PRINT Display:	Z80 Opcode:
94	148	/G/	STOP	/G/	SUB H
95	149	/G/	ELSE	/G/	SUB L
96	150	/G/	TRON	/G/	SUB (HL)
97	151	/G/	TROFF	/G/	SUB A
98	152	/G/	DEFSTR	/G/	SBC A,B
99	153	/G/	DEFINT	/G/	SBC A,C
9A	154	/G/	DEFSNG	/G/	SBC A,O
9B	155	/G/	DEFDBL	/G/	SBC A,E
9C	156	/G/	LINE	/G/	SBC A,H
9D	157	/G/	EQIT	/G/	SBC A,L
9E	158	/G/	ERROR	/G/	SBC A,(HL)
9F	159	/G/	RESUME	/G/	SBC A,A
AD	160	/G/	OUT	/G/	AND B
A1	161	/G/	ON	/G/	AND C
A2	162	/G/	OPEN	/G/	AND O
A3	163	/G/	FIELD	/G/	AND E
A4	164	/G/	GET	/G/	AND H
A5	165	/G/	PUT	/G/	AND L
A6	166	/G/	CLOSE	/G/	AND (HL)
A7	167	/G/	LOAD	/G/	AND A
AB	168	/G/	MERGE	/G/	XOR B
A9	169	/G/	NAME	/G/	XOR C
AA	170	/G/	KILL	/G/	XOR O
AB	171	/G/	LSET	/G/	XOR E
AC	172	/G/	RSET	/G/	XOR H
AD	173	/G/	SAVE	/G/	XOR L
AE	174	/G/	SYSTEM	/G/	XOR (HL)
AF	175	/G/	LPRINT	/G/	XOR A
B0	176	/G/	DEF	/G/	OR B
B1	177	/G/	POKE	/G/	OR C
B2	178	/G/	PRINT	/G/	OR O
B3	179	/G/	CONT	/G/	OR E
B4	180	/G/	LIST	/G/	OR H
B5	181	/G/	LLIST	/G/	OR L
B6	182	/G/	DELETE	/G/	OR (HL)
B7	183	/G/	AUTO	/G/	OR A
B8	184	/G/	CLEAR	/G/	CP B
B9	185	/G/	CLOAD	/G/	CP C
BA	186	/G/	CSAVE	/G/	CP O
BB	187	/G/	NEW	/G/	CP E
BC	188	/G/	TAB(	/G/	CP H
BD	189	/G/	TO	/G/	CP L
BE	190	/G/	FN	/G/	CP (HL)
BF	191	/G/	USING	/G/	CP A
C0	192	/G/	VARPTR	/G/	RET NZ
C1	193	/G/	USR	TAB+01	POP BC
C2	194	/G/	ERL	TAB+02	JP NZ,NN
C3	195	/G/	ERR	TAB+03	JP NN
C4	196	/G/	STRING\$	TAB+04	CALL NZ,NN
C5	197	/G/	INSTR	TAB+05	PUSH BC
C6	198	/G/	POINT	TAB+06	ADD A,N
C7	199	/G/	TIME\$	TAB+07	RST 00H

---

Upper Character Set

---

HEX Val	DEC Val	POKE Display:	BASIC Keyword:	PRINT Display:	Z80 Opcode:
C8	200	/G/	MEM	TAB+08	RET Z
C9	201	/G/	INKEY\$	TAB+09	RET
CA	202	/G/	THEN	TAB+10	JP Z, NN
CB	203	/G/	NOT	TAB+11	<Note 1>
CC	204	/G/	STEP	TAB+12	CALL Z, NN
CD	205	/G/	+	TAB+13	CALL NN
CE	206	/G/	-	TAB+14	ADC A, N
CF	207	/G/	*	TAB+15	RST 08H
D0	208	/G/	/	TAB+16	RET NC
D1	209	/G/	**	TAB+17	POP DE
D2	210	/G/	AND	TAB+18	JP NC, NN
D3	211	/G/	OR	TAB+19	OUT (N), A
D4	212	/G/	>	TAB+20	CALL NC, NN
D5	213	/G/	=	TAB+21	PUSH OE
D6	214	/G/	<	TAB+22	SUB N
D7	215	/G/	SGN	TAB+23	RST 10H
D8	216	/G/	INT	TAB+24	RET C
D9	217	/G/	ABS	TAB+25	EXX
DA	218	/G/	FRE	TAB+26	JP C, NN
DB	219	/G/	INP	TAB+27	IN A, (N)
DC	220	/G/	POS	TAB+28	CALL C, NN
DD	221	/G/	SQR	TAB+29	<Note 2>
DE	222	/G/	RND	TAB+30	SBC A, N
DF	223	/G/	LOG	TAB+31	RST 1BH
E0	224	/G/	EXP	TAB+32	RET PO
E1	225	/G/	COS	TAB+33	POP HL
E2	226	/G/	SIN	TAB+34	JP PO, NN
E3	227	/G/	TAN	TAB+35	EX (SP), HL
E4	228	/G/	ATN	TAB+36	CALL PO, NN
E5	229	/G/	PEEK	TAB+37	PUSH HL
E6	230	/G/	CVI	TAB+38	AND N
E7	231	/G/	CVS	TAB+39	RST 20H
E8	232	/G/	CVD	TAB+40	RET PE
E9	233	/G/	EOF	TAB+41	JP (HL)
EA	234	/G/	LOC	TAB+42	JP PE, NN
EB	235	/G/	LOF	TAB+43	EX OE, HL
EC	236	/G/	MKI\$	TAB+44	CALL PE, NN
ED	237	/G/	MKS\$	TAB+45	<Note 3>
EE	238	/G/	MKO\$	TAB+46	XOR N
EF	239	/G/	CINT	TAB+47	RST 2BH
FO	240	/G/	CSNG	TAB+48	RET P
F1	241	/G/	COBL	TAB+49	POP AF
F2	242	/G/	FIX	TAB+50	JP P, NN
F3	243	/G/	LEN	TAB+51	OI
F4	244	/G/	STR\$	TAB+52	CALL P, NN
F5	245	/G/	VAL	TAB+53	PUSH AF
F6	246	/G/	ASC	TAB+54	OR N
F7	247	/G/	CHR\$	TAB+55	RST 30H
F8	248	/G/	LEFT\$	TAB+56	RET M
F9	249	/G/	RIGHT\$	TAB+57	LD SP, HL
FA	250	/G/	MIO\$	TAB+58	JP M, NN
FB	251	/G/	_____	TAB+59	EI

Upper Character Set					
HEX Val	DEC Val	POKE Display:	BASIC Keyword:	PRINT Display:	Z80 Opcode:
FC	252	/G/	---	TAB+60	CALL M,NN
FD	253	/G/	---	TAB+61	<Note 4>
FE	254	/G/	---	TAB+62	CP N
FF	255	/G/	---	TAB+63	RST 38H

Note 1: CB includes RLC, RRC, RL, RR, SLA, SRL, BIT, RES, SET.

Note 2: DD includes IX register manipulations.

Note 3: ED includes miscellaneous IN, OUT, LD, and ADC functions as well as block move and compare commands, and interrupt mode setting and handling.

Note 4: FD includes IY register manipulations.

Graphics Note: In the table above /G/ means a graphics character will be displayed. The table below shows the graphics characters and their ASCII code in decimal.

128	129	130	131
132	133	134	135
136	137	138	139
140	141	142	143
144	145	146	147
148	149	150	151
152	153	154	155
156	157	158	159
160	161	162	163
164	165	166	167
168	169	170	171
172	173	174	175
176	177	178	179
180	181	182	183
184	185	186	187
188	189	190	191

```

00100 ;
00110 ;
00120 ; ***** K E E P I T   3 . 2   *****
00130 ;
00140 ; >>>> SYSTEM /12345 TO ENTER <<<<
00150 ;
00160 ; >>>> USE *READ FOR MENU <<<<
00170 ;
00180 ;
00190 ; COPYRIGHT (C) 1980, 1981 BY DENNIS BATHORY KITSZ
00200 ; ALL RIGHTS RESERVED. NO PART OF THIS PROGRAM MAY
00210 ; BE REPRODUCED BY ANY MEANS, ELECTRONIC, ELECTRO-
00220 ; MECHANICAL, OR IN PRINT, WHETHER BY METHODS IN
00230 ; USE OR CONCEIVED IN THE FUTURE, WITHOUT SPECIFIC
00240 ; WRITTEN PERMISSION OF THE AUTHOR.
00250 ;
00260 INKEYS EQU 4099H ;INKEY$ BYTE STORAGE AREA
00270 KYSCAN EQU 03F8H ;KEYBOARD SCAN ROUTINE
00280 PORTFF EQU 4030H ;CASSETTE OUTPUT PORT
00290 KPLACE EQU 401AH ;1-BYTE KEYSTROKE STORE
00300 SHIFTR EQU 4018H ;STORAGE FOR LC DRIVER
00310 RESTRT EQU 4004H ;BASIC INTERP PATCH POINT
00320 STGEN0 EQU 40ADH ;END OF STRINGS POINTER
00330 STACKR EQU 40E8H ;BASIC STACK POINTER
00340 VARENO EQU 40FDH ;END OF VARIABLE POINTER
00350 READY EQU 06CCH ;RETURN TO READY INTACT
00360 MEMTOP EQU 4081H ;TOP OF BASIC MEMORY
00370 SETPTS EQU 4000H ;SETPOINT RESTART ADDRESS
00380 WRT8YT EQU 0264H ;ROM BYTE WRITE ROUTINE
00390 8YTE EQU 1078H ;ROM READ KEYS & TOKENIZE
00400 VIDEO EQU 3C00H ;FIRST SCREEN LOCATION
00410 ;
3039 00420 ORG 3039H
00430 ; PREPARE RAM AREAS FOR USE
00440 START LD HL,(4081H)
00450 LD DE,(40ADH)
00460 LD 8,7
00470 OEC HL
00480 OEC DE
00490 DJNZ $-2
00500 LD (4081H),HL
00510 LD (40ADH),DE
00520 INC HL
00530 PUSH HL
00540 POP IY
00550 ; READY INTERPRETER VECTOR PATCH
00560 ; GET CURRENT CONTENTS OF 4004H
00570 LD A,0C3H
00580 LD (IY+0),A
00590 LD 0E,(RESTRT)
00600 LD (IY+1),E
00610 LD (IY+2),0
00620 LD (IY+3),A
00630 LD 0E,(SKPSTP)
00640 LD (IY+4),E
00650 LD (IY+5),0
00660 GRINS INC HL
00670 INC HL
00680 INC HL
00690 LD (RESTRT),HL
00700 XOR A
00710 LD (SHIFTR),A
00720 ; CALL CLEAR AND RETURN TO BASIC
00730 RSLVII CALL 01C9H
00740 CALL 1661H
00750 LD HL,0111H
00760 CALL 28A7H
00770 JP READY
00780 ; START KEYBOARD SCAN
00790 KBPFIX LD HL,4036H
00800 LD 8C,3801H
00810 LD 0,0
00820 ; CHECK EACH ROW OF KEYS
00830 KEYPRS LD A,(8C)
00840 LD E,A
00850 AND E
00860 JR NZ,STROKE
00870 LD (HL),A
00880 ; INC AND ROTATE TO CHECK NEXT ROW
00890 RECHKE INC 0
00900 INC L
00910 RLC C
00920 LD A,C
00930 ; CHECK IF LAST ROW (NOT INCL. SHIFT)
00940 SUB 80H
00950 JR NZ,KEYPRS
00960 ; CHECK IF KEYBOARD CLEAR
00970 LD 8,7
00980 CLRMEM OEC L
00990 ADO A,(HL)
00A00 DJNZ CLRMEM
00A10 AND A

```

```

30A3 3E00 01020 LD A,0
30A5 C0 01030 RET NZ
01040 ; RESET AUTOREPEAT DELAY TO 0
01050 LD (KPLACE),A
30A6 321A40 01060 RET
30A9 C9 01070 ; IF KEYSTROKE FOUND CHECK AUTOREPEAT LOOP
01080 STROKE AND (HL)
30AA A6 01090 JR Z,FOUND
30AB 281E 01100 LD A,(INKEYS)
30AD 3A9940 01110 AND A
30B0 A7 01120 JR NZ,RECHKE
30B1 20E0 01130 LD A,(KPLACE)
30B3 3A1A40 01140 INC A
30B6 3C 01150 LD (KPLACE),A
30B7 321A40 01160 CP OFFH
30BA FEFF 01170 JR Z,DECA
30BC 2808 01180 PUSH 8C
30BE C5 01190 LD 8,OFFH
30BF 06FF 01200 TWMSTE DJNZ TWMSTE
30C1 10FE 01210 POP 8C
30C3 C1 01220 JR RECHKE
30C4 18C0 01230 OEC A
30C6 30 01240 LD (KPLACE),A
30C7 321A40 01250 ; GET AND SAVE FOUND KEYBOARD BYTE
01260 LD A,E
30CA 78 01270 FOUND LD (HL),E
30CB 73 01280 LD A,0
30CC 7A 01290 RLCA
30CD 07 01300 RLCA
30CE 07 01310 RLCA
30CF 07 01320 LD 0,A
30D0 57 01330 LD C,1
30D1 0E01 01340 BACKUP LD A,C
30D3 79 01350 AND E
30D4 A3 01360 JR NZ,AROUND
30D5 2005 01370 INC 0
30D7 14 01380 INC C
30D8 C801 01390 JR BACKUP
30DA 18F7 01400 AROUND LD A,(3290H)
30DC 3A8038 01410 LD 8,A
30DE 47 01420 LD A,0
30E0 7A 01430 AOD A,40H
30E1 C640 01440 CP 60H
30E3 FE60 01450 JR NC,Z0429H
30E5 3016 01460 LD 0,A
30E7 57 01470 LD A,(3840H)
30E8 3A4038 01480 AND 10H
30EA E610 01490 JR NZ,CNTRL
30EC 2009 01500 LD A,0
30EE 7A 01510 RRC 8
30F0 C808 01520 JR C,GOAWAY
30F2 3830 01530 AOD A,20H
30F4 C620 01540 JR GOAWAY
30F6 1839 01550 LD A,0
30F8 7A 01560 SUB 40H
30FA 0640 01570 JR GOAWAY
30FB 1834 01580 SUB 70H
30FD 0670 01590 JR NC,Z0430H
30FF 3010 01600 ADD A,40H
3101 C640 01610 CP 3CH
3103 FE3C 01620 JR C,Z0435H
3105 3802 01630 XOR 10H
3107 EE10 01640 Z0435H RRC 8
3109 C808 01650 JR NC,GOAWAY
310B 3024 01660 XOR 10H
310D EE10 01670 JR GOAWAY
310F 1820 01680 Z0430H RLCA
3111 07 01690 RRC 8
3112 C808 01700 JR NC,Z0443H
3114 3001 01710 LD A
3116 3C 01720 INC A
3117 212131 01730 LD HL,TABLET
311A 4F 01740 LD C,A
311B 0600 01750 LD 8,0
311D 09 01760 AOD HL,8C
311E 7E 01770 LD A,(HL)
311F 1810 01780 JR GOAWAY
3121 0000 01790 TABLET DEFW 0000H
3123 1F1F 01800 DEFW 1F1FH
3125 0101 01810 DEFW 0101H
3127 5818 01820 DEFW 185BH
3129 DA00 01830 DEFW 000AH
312B 0818 01840 DEFW 180BH
312D 0919 01850 DEFW 1909H
312F 2020 01860 DEFW 2020H
3131 57 01870 GOAWAY LD 0,A
3133 3A1038 01880 LD A,(3810H)
3135 FE01 01890 CP 1
3137 2016 01900 JR NZ,BLEEP
3139 3A8038 01910 LD A,(3880H)
313B FE01 01920 CP 1
313D 200F 01930 JR NZ,BLEEP
3140 3A1940 01940 LD A,(SHIFTR)
3143 EE01 01950 XOR 1

```

278

32B8 CDC032	03820	CALL	OUTSEQ	04750 ; DISPLAY ASCII VALUES TOO	
	03830 ; DUMP COMPLETE VIDEO MEMORY			04760 LD B,10H	
32B8 21003C	03840	LD	HL,VIDEO	04770 PUSH BC	
329F D604	03850	LD	B,4	04780 DEC DE	
32A1 C5	03860 COEF	PUSH	BC	04790 DJNZ \$-1	
32A2 C0B832	03870	CALL	PREQUT	04800 POP BC	
32A5 C1	03880	POP	BC	04810 PUSH BC	
32A6 10F9	03890	DJNZ	COEF	04820 LD HL,3CC0H	
	03900 ; WRITE END OF PROGRAM CODE (7B)			04830 LD A,(DE)	
32A8 3E78	03810	LD	A,7BH	04840 LD (HL),A	
32AA C06402	03820	CALL	WRTBYT	04850 INC HL	
	03930 ; WRITE START ADDRESS AFTER LOAD (06CCH)			04860 INC HL	
32A0 3ECC	03840	LD	A,DCCH	04870 INC HL	
32AF C06402	03850	CALL	WRTBYT	04880 INC DE	
32B2 3E06	03860	LD	A,06	04890 DJNZ BBBA	
32B4 C06402	03870	CALL	WRTBYT	04900 POP BC	
	03880 ; RESTORE BASIC INFORMATION TO REGISTERS			04910 DEC DE	
32B7 C1	03890	POP	BC	04920 DJNZ \$-1	
32B8 01	04000	POP	DE	04930 RET	
32B9 F1	04010	POP	AF	04940 ; SCAN FOR EDIT / GET THIRO SCREEN LINE	
32BA E1	04020	POP	HL	04950 NEXT99 CALL CONTNT	
	04030 ; RETURN TO BASIC PROGRAM IN PROGRESS			04960 ; SCAN KEYBOARD FOR BREAK, ARROWS	
32B8 C3CC06	04040	JP	READY	04970 EDITOR LD A,(3B40H)	
	04050 ; OUTPUT SEQUENCE SUBROUTINE			04980 RLA	
	04060 ; WRITE BLDCK HEADER CODE (3C)			04990 RLA	
32BE D600	04070 PREOUT	LD	B,0	05000 JR NC,AAAA	
32C0 3E3C	04080 OUTSEQ	LD	A,3CH	05010 INC DE	
32C2 C06402	04080	CALL	WRTBYT	05020 JR STNDRD	
	04100 ; GET NUMBER OF BYTES TO WRITE			05030 AAAAA RLA	
32C5 78	04110	LD	A,B	05040 JR NC,AAAB	
32C6 C06402	04120	CALL	WRTBYT	05050 DEC DE	
	04130 ; GET START ADDRESS LSB, SAVE IN C (CHECKSUM)			05060 JR STNDRD	
32C9 70	04140	LD	A,L	05070 LD B,10H	
32CA 4F	04150	LD	C,A	05080 RLA	
32CB C06402	04160	CALL	WRTBYT	05090 JR NC,AAAC	
	04170 ; GET START ADDRESS MSB, SAVE IN C (CHECKSUM)			05100 DEC DE	
32CE 79	04180	LD	A,C	05110 DJNZ \$-1	
32CF B4	04190	ADD	A,H	05120 STNDRD	
32D0 4F	04200	LD	C,A	05130 AAAC RLA	
32D1 7C	04210	LD	A,H	05140 JR NC,BREEK	
32D2 C06402	04220	CALL	WRTBYT	05150 INC DE	
	04230 ; GET BLOCK OF DATA, WRITE, AND SAVE IN C			05160 DJNZ \$-1	
32D5 78	04240	LD	A,C	05170 JR STNDRD	
32D6 B6	04250	ADD	A,(HL)	05180 BREEK RLA	
32D7 4F	04260	LD	C,A	05190 JR NC,AAAD	
32D8 7E	04270	LD	A,(HL)	05200 POP HL	
32D9 C06402	04280	CALL	WRTBYT	05210 JP READY	
32DC 23	04290	INC	HL	05220 ; DISPLAY EDITING / GET FIFTH SCREEN LINE	
32D0 10F6	04300	DJNZ	WRTPGE	05230 AAAD LD HL,3001H	
	04310 ; GET CHECKSUM FROM C AND WRITE TO TAPE			05240 LD (HL),5FH	
32DF 78	04320	LD	A,C	05250 DEC HL	
32E0 C06402	04330	CALL	WRTBYT	05260 LD (HL),5FH	
32E3 C8	04340	RET		05270 LD B,2	
	04350 ; GET REST OF DATA AND CONVERT			05280 AAAP PU5H DE	
32E4 C07B10	04360 OPENER	CALL	BYTE	05290 PU5H HL	
32E7 FE22	04370	CP	22H	05300 CALL 0049H	
32E8 C2B831	04380	JP	NZ,SYNERR	05310 POP HL	
32EC E5	04390	PUSH	HL	05320 POP DE	
32E0 C0D133	04400	CALL	XX99	05330 CP 47H	
32F0 CDC801	04410	CALL	01C9H	05340 JR NC,EDITOR	
32F3 1B40	04420	JR	NEXT99	05350 CP 30H	
	04430 ; GET 18 SCREEN POSITIONS READY (10H)			05360 JR C,EDITOR	
32F5 7A	04440 CONTNT	LD	A,0	05370 CP 3AH	
32F6 21403C	04450	LD	HL,3C40H	05380 JR C,AAAF	
32F9 E6F0	04460	AND	0F0H	05390 CP 40H	
32FB C0B833	04470	CALL	RRRRS	05400 JR C,EDITOR	
32FE 7A	04480	LD	A,0	05410 AAAP LD (HL),A	
32FF E60F	04490	AND	0FH	05420 INC HL	
3301 C0B033	04500	CALL	HEXASC	05430 DJNZ AAAP	
3304 77	04510	LD	(HL),A	05440 ; CONVERT CHOSEN DATA TO HEX	
3305 23	04520	INC	HL	05450 DEC HL	
3306 78	04530	LD	A,E	05460 CALL ASCHEX	
3307 E6F0	04540	AND	0F0H	05470 LD C,A	
3308 C0B833	04550	CALL	RRRRS	05480 DEC HL	
330C 78	04560	LD	A,E	05490 CALL LLLLS	
330D E60F	04570	AND	0FH	05500 ADD A,C	
330F C0B033	04580	CALL	HEXASC	05510 ; PUT NEW BYTE IN PLACE	
3312 77	04590	LD	(HL),A	05520 LD (DE),A	
3313 21B03C	04600	LD	HL,3C80H	05530 INC DE	
3316 D610	04610	LD	B,10H	05540 ; DISPLAY REVISED LINE OF DATA	
	04620 ; DISPLAY CONTENTS OF ADDRESS CHOSEN			05550 STNDRD CALL CONTNT	
3318 1A	04630 CONT02	LD	A,(DE)	05560 CALL DELAY	
3319 E6F0	04640	AND	0F0H	05570 JR EDITOR	
331B C0B833	04650	CALL	RRRRS	05580 ; ASCII TO HEXADECIMAL CONVERSION	
331E 1A	04660	LD	A,(DE)	05590 ASCHEX LD A,(HL)	
331F E60F	04670	AND	0FH	05600 SUB 30H	
3321 C0B033	04680	CALL	HEXASC	05610 CP 0AH	
3324 77	04690	LD	(HL),A	05620 RET C	
3325 23	04700	INC	HL	05630 SUB 7	
3326 23	04710	INC	HL	05640 RET	
3327 13	04720	INC	DE	05650 ; HEXADECIMAL TO ASCII CONVERSION	
	04730 ; REDO IT 16 TIMES FOR FULL LINE			05660 HEXASC ADD A,30H	
332B 10EE	04740	DJNZ	CONT02	05670 CP 3AH	
332A D610	04750				
332C C5	04760				
332D 18	04770				
332E 10F0	04780				
3330 C1	04790				
3331 C5	04800				
3332 21C03C	04810				
3335 1A	04820				
3336 77	04830				
3337 23	04840				
3338 23	04850				
3339 23	04860				
333A 13	04870				
333B 10FB	04880				
333C C1	04890				
333E 18	04900				
333F 10F0	04910				
3341 C9	04920				
	04930				
	04940 ; SCAN FOR EDIT / GET THIRO SCREEN LINE				
3342 CDF532	04950 NEXT99	CALL	CONTNT		
	04960 ; SCAN KEYBOARD FOR BREAK, ARROWS				
3345 3A403B	04970 EDITOR	LD	A,(3B40H)		
3348 17	04980	RLA			
3349 17	04990	RLA			
334A 3003	05000	JR	NC,AAAA		
334C 13	05010	INC	DE		
334D 1850	05020	JR	STNDRD		
334F 17	05030	AAAA	RLA		
3350 3003	05040	JR	NC,AAAB		
3352 18	05050	DEC	DE		
3353 184A	05060	JR	STNDRD		
3355 0610	05070	AAAB	LD B,10H		
3357 17	05080	RLA			
3358 3005	05090	JR	NC,AAAC		
335A 18	05100	DEC	DE		
335B 10F0	05110	DJNZ	\$-1		
335D 1840	05120	STNDRD			
335F 17	05130	AAAC	RLA		
3360 3005	05140	JR	NC,BREEK		
3362 13	05150	INC	DE		
3363 10F0	05160	DJNZ	\$-1		
3365 183B	05170	JR	STNDRD		
3367 17	05180	BREEK	RLA		
3368 3004	05190	JR	NC,AAAD		
336A E1	05200	POP	HL		
336B C3CC06	05210	JP	READY		
	05220 ; DISPLAY EDITING / GET FIFTH SCREEN LINE				
336E 210130	05230 AAAD	LD	HL,3001H		
3371 365F	05240	LD	(HL),5FH		
3373 2B	05250	DEC	HL		
3374 365F	05260	LD	(HL),5FH		
3376 D602	05270	LD	B,2		
337B 05	05280	AAAP	PU5H DE		
3379 E5	05290	PU5H	HL		
337A C04900	05300	CALL	0049H		
337D E1	05310	POP	HL		
337E 01	05320	POP	DE		
337F FE47	05330	CP	47H		
3381 30C2	05340	JR	NC,EDITOR		
3383 FE30	05350	CP	30H		
3385 38BE	05360	JR	C,EDITOR		
3387 FE3A	05370	CP	3AH		
3389 3804	05380	JR	C,AAAF		
338B FE40	05390	CP	40H		
338D 38BE	05400	JR	C,EDITOR		
338F 77	05410	AAAP	LD (HL),A		
3390 23	05420	INC	HL		
3391 10E5	05430	DJNZ	AAAP		
	05440 ; CONVERT CHOSEN DATA TO HEX				
3393 2B	05450	DEC	HL		
3394 CDA733	05460	CALL	ASCHEX		
3397 4F	05470	LD	C,A		
3398 2B	05480	DEC	HL		
3399 CDC233	05490	CALL	LLLLS		
339C B1	05500	ADD	A,C		
	05510 ; PUT NEW BYTE IN PLACE				
339D 12	05520	LD	(DE),A		
339E 13	05530	INC	DE		
	05540 ; DISPLAY REVISED LINE OF DATA				
339F CDF532	05550 STNDRD	CALL	CONTNT		
33A2 CDCA33	05560	CALL	DELAY		
33A5 189E	05570	JR	EDITOR		
	05580 ; ASCII TO HEXADECIMAL CONVERSION				
33A7 7E	05590	ASCHEX	LD A,(HL)		
33AB D630	05600	SUB	30H		
33AA FEDA	05610	CP	0AH		
33AC D8	05620	RET	C		
33AD D607	05630	SUB	7		
33AF C9	05640	RET			
	05650 ; HEXADECIMAL TO ASCII CONVERSION				
33B0 C630	05660	HEXASC	ADD A,30H		
33B2 FE3A	05670	CP	3AH		

280

```

34B7 C3CC06 07540 PREP2 JP READY
34BA 07550 ; THIS IS THE STEPPER ROUTINE
34BA F5 07560 JMPPOS EQU $
34BB C5 07570 STEPPR PUSH AF
34BB C5 07580 PUSH BC
34BB C5 07590 ; WAIT FOR SHIFT TO BE PRESSED
34BC 3A803B 07600 LUPER LD A,(3B80H)
34BF A7 07610 AND A
34C0 28FA 07620 JR Z,LUPER
34C0 28FA 07630 ; LDA0 DELAY VALUE INTO BC
34C2 4F 07640 LD C,A
34C3 F04606 07650 LD B,(IY+6)
34C3 F04606 07660 ; CALL DELAY IN ROM
34C6 C06000 07670 CALL 0600H
34C9 C1 07680 POP BC
34CA F1 07690 POP AF
34C9 C1 07700 ; BACK TO REST OF TEST SEQUENCE
34CB C37631 07710 JP BEGIN
34CB C37631 07720 ; BEGIN MEMORY RESET SEQUENCE
34CB C37631 07730 ; CHECK FOR QUOTE MARK DELIMITER
34CE C07B10 07740 MEMSET CALL 8YTE
34D1 FE22 07750 CP 22H
34D3 20CE 07760 JR NZ,SYN3
34D3 20CE 07770 ; CONVERT *MEM OPERAND TO HEX
34D5 C00133 07780 CALL XX89
34D5 C00133 07790 ; CHECK FOR >4400H MEMORY ADDRESS
34D5 C00133 07800 ; GO TO ON ERROR IF NOT ENOUGH
34D8 210044 07810 LD HL,4400H
34D8 AF 07820 XOR A
34DC E052 07830 SBC HL,0E
34DE 027A19 07840 JP NC,197AH
34DE 027A19 07850 ; TEST FOR MEMORY RESET <F8BBH
34DE 027A19 07860 ; ON ERROR IF TOO MUCH
34E1 05 07870 PUSH OE
34E2 E1 07880 POP HL
34E3 D607 07890 LD B,7
34E5 23 07900 INC HL
34E5 10F0 07910 DJNZ $-1
34E8 7E 07920 LD A,(HL)
34E9 47 07930 LO B,A
34EA 2F 07940 CPL
34E8 77 07950 LD (HL),A
34EC 8E 07960 CP (HL)
34ED C27A19 07970 JP NZ,197AH
34F0 70 07980 LD (HL),B
34F0 70 07990 ; PUT NEW MEMORY SIZE INTO PLACE
34F1 E053B140 08000 LD (40B1H),DE
34F1 E053B140 08010 ; TRANSFER KEEPIT DATA ROW
34F1 E053B140 08020 ; AND PUT NEW ADDRESS IN IY
34F5 F0E5 08030 PUSH IY
34F7 E1 08040 POP HL
34F8 13 08050 INC OE
34F9 05 08060 PUSH OE
34FA F0E1 08070 POP IY
34FC D10700 08080 LD BC,7
34FF E080 08090 LDIR
3501 F0E5 08100 PUSH IY
3503 E1 08110 POP HL
3503 E1 08120 ; MAKE FRE(A$) MEM SIZE - 50 DECIMAL
3504 D632 08130 LD B,32H
3506 18 08140 STRING DEC OE
3507 10F0 08150 DJNZ STRING
3507 10F0 08160 ; PUT NEW STRING POINTER IN PLACE
3509 E053A040 08170 LD (40A0H),OE
3500 C36C30 08180 JP GRINS
3500 C36C30 08190 ; THIS IS LOWER CASE DETERMINATION
3510 F5 08200 LOWER PUSH AF
3511 3A1940 08210 LD A,(5HIFTR)
3514 FE01 08220 CP 1
3516 2B04 08230 JR Z,LOWER1
351B F1 08240 POP AF
3519 C35B04 08250 JP 045BH
351C F1 08260 LOWER1 POP AF
3510 006E03 08270 LD L,(IX+3)
3520 006604 08280 LD H,(IX+4)
3523 DA9A04 08290 JP C,049AH
3526 007E05 08300 LD A,(IX+5)
3529 B7 08310 OR A
352A 2B01 08320 JR Z,GETCHR
352C 77 08330 LD (HL),A
352D 79 08340 GETCHR LD A,C
352E FE20 08350 CP 20H
3530 DA0605 08360 JP C,0506H
3533 FE80 08370 CP 80H
3535 02A604 08380 JP NC,04A6H
3538 C37004 08390 JP 047DH
3538 2A 08400 INTRO1 OEFM '*FIX = SET DEBOUNCE/BEEP/AUTOREPEAT'
355E 0A 08410 OEFB OAH
355F 2A 08420 OEFM '*KILL = RESET TO NORMAL KEYBOARD'
357F 0A 08430 OEFB OAH
3580 2A 08440 OEFM '*MEM = RESET MEMORY SIZE (NOTE 1)'
35A1 00 08450 OEFB 00H
35A2 00 08460 OEFB 00H

35A3 2A 08470 INTRO2 OEFM '*PRINT = LOWERCASE KEYBOARD/DISPLAY ON'
35C8 0A 08480 OEFB OAH
35CA 2A 08490 OEFM '*OUT = LOWERCASE KEYBOARD/DISPLAY OFF'
35EF 0A 08500 OEFB OAH
35F0 2A 08510 OEFM '*NEW = RESTORE PROGRAM VICTIM OF NEW'
3614 0A 08520 OEFB OAH
3615 2A 08530 OEFM '*OPEN = HEX/ASCII/GRAPHICS MONITOR (NOTE 1)'
3640 00 08540 OEFB 00H
3641 00 08550 OEFB 00H
3642 2A 08560 INTRO3 OEFM '*SAVE/RUN = SAVE RUNNING PROGRAM (NOTE 2)'
366B 0A 08570 OEFB OAH
366C 2A 08580 OEFM '*SAVE/OPEN = MEMORY BLOCK SAVE (NOTE 3)'
3693 0A 08590 OEFB OAH
3694 2A 08600 OEFM '*STEPXX = SINGLE STEPPER ON, XX = DELAY'
36B8 0A 08610 OEFB OAH
36BC 2A 08620 OEFM '*STEP = SINGLE STEPPER OFF'
3806 00 08630 OEFB 00H
3607 00 08640 OEFB 00H
3608 4E 08650 INTRO4 OEFM 'NOTE 1. REQUIRES 4-CHARACTER HEX VALUE IN QUOTES.'
370A 0A 08660 OEFB OAH
3708 4E 08670 OEFM 'NOTE 2. REQUIRES 6-CHARACTER NAME IN QUOTES.'
3738 0A 08680 OEFB OAH
3738 4E 08690 OEFM 'NOTE 3. SAME AS ABOVE PLUS HEX START,
3773 00 08700 OEFB 00H
3774 00 08710 OEFB 00H
3038 08720 ENO START
00000 TOTAL ERRORS
18637 TEXT AREA BYTES LEFT

AAAA 334F 05030 05000
AAAB 3355 05070 05040
AAAC 335F 05130 05090
AAAD 336E 05230 05190
AAAE 3378 05280 05430
AAAF 33BF 05410 05380
ABCO 34A6 07430 07380
AROUNO 300C 01400 01360
ASCHEX 33A7 05590 05460 05810 05990 06080
BACKUP 3003 01340 01390
88BA 3335 04830 04890
8EEEEP 3132 01870
BEEPER 3163 02120 02200
BEGIN 3176 02260 07710
BIPOFF 31F0 02850 02580
BLEEEP 314F 02000 01890 01920
BREEK 3367 05180 05140
8YTE 107B 00390
02330 02350 02400 03020 03050 03090 03130
03310 04360 05940 06210 06330 06450 06520
06620 07180 07360 07740

COEF 32A1 03860 03890
CLRMEM 308E 00880 01000
CNTRL 30FB 01650 01490
CONTO2 331B 04630 04740
CONTNT 32F5 04440 04950 05550
OECA 30C5 01230 01170
DELAY 33CA 05880 05560
DUMP 325A 03420 03360
DUMP99 3422 06450 06380
EDITOR 3345 04970 05340 05360 05400 05570
EXECU 344C 06700 06640
FGHIJ 3213 03000
FINSH1 327C 03630 03590
FOUNO 30CB 01270 01090
FREQUY 316C 02180 02180
GETCHR 3520 08340 08320
GDaway 3131 01860 01520 01540 01570 01650 01670 01770
GRIN5 306C 00660 08180
HEXASC 33B0 05660 04500 04580 04680 05760
IIII 3494 07290 07210
INBEEP 31FB 02890 02540
INKEYS 4099 00260 01100
INTRO1 353B 08400 02650
INTRO2 35A3 08470 02670
INTRO3 3642 08560 02690
INTRO4 360B 08650 02710
JMPPOS 34BA 07560 07470 07480
KBPFIX 30B5 00790 02890
KEEPIT 3294 03800 03760
KEYPRS 3080 00830 00950
KPLACE 401A 00290 01050 01130 01150 01240
KYSCAN 03F8 00270
LLLLS 33C2 05810 05490 06030 06120
LOWCAS 3108 02750 02560
LOWER 3510 08200 02770
LOWER1 351C 08260 08230
LP99 3465 06980 07030
LUPER 34BC 07600 07620
MACH 33F4 06180 03070
MEMSET 34CE 07740 02520
MEMTOP 40B1 00360 03650
MENU 31C1 02650 02620
NAME99 3400 06330 06370

```

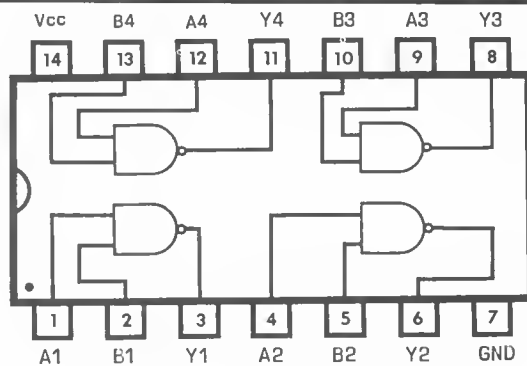
```

NAMES 3245 03310 03350
NEXTB9 3342 04950 04420
NEXTBT 3253 03380 03330 03400
NOTROY 317F 02320 02290
NXTB99 3418 06410 06350 06430
NXTBCH 328A 03730 03780
NXTPGE 3272 03560 03610
OKSTAR 3180 02400 02370
OPENER 32E4 04360 02480
OUTSEQ 32C0 04080 03430 03440 03470 03540 03600 03710 03770
03820 06990
PORTFF 4030 00280 02070
PREOUT 328E 04070 03870
PREP2 3487 07540 07130 07270
READY 06CC 00350 00770 02910 04040 05210 07540
READYX 31FE 02910 02730 02790 02830 02870
RECEK 3083 00890 01120 01220
RENEW 3201 02930 02460
RESTR 4004 00310 00590 00690
RRRRS 3388 05720 04470 04550 04650
RSLVII 3076 00730 03000
SAVE 3183 02430
SAVER 3216 03020 02440
SETPTS 4000 00370 03420
SHIFTR 4018 00300 00710 01930 01950 02760 08210
SKPSTP 3176 02250 00630 07230 07240
SSSS 3303 05940 05960
STACKR 40E8 00330 03630
START 3039 00440 08720
STEPPR 348A 07570
STGEN0 40A0 00320
STNDRO 339F 05550 05020 05060 05120 05170
STPSET 3480 07160 02500
STRING 3506 08140 08150
STROKE 30AA 01080 00860
SYN2 33F0 06230 06180 06470
SYN3 34A3 07410 07760
SYNERR 318E 02630 02410 03040 03110 04380 06230 07410
TABLET 3121 01780 01720
TMWSTE 30C1 01200 01200
TTTT 3439 06590 06540
UPPPER 31E8 02810 02600
VARENO 40F0 00340 03490
VIDEO 3C00 00400 03840
WRIT99 3453 06740 06680
WRTBYT 0264 00380 03270 03340 03390 03920 03850 03970 04090
04120 04160 04220 04280 04330 06290 06360
06420 07060 07090 07110
04300
WRTPGE 3205 04240 04400 06490 06590 06700 07780
XXB9 3301 05930 01450
Z0428H 30F0 01580 01620
Z0435H 3109 01640 01590
Z0430H 3111 01680 01700
Z0443H 3117 01720

```

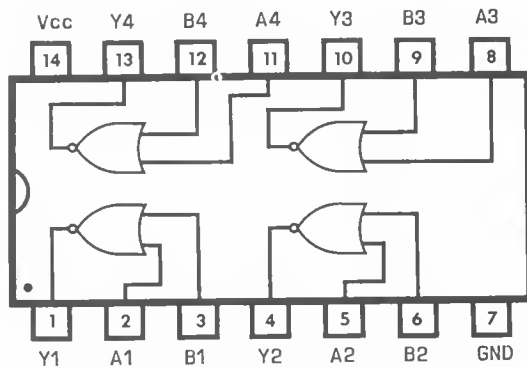
KEEPIT 3.2 is a 2K utility program created to extend the capabilities of Level II BASIC. It was originally sold by The Alternate Source in a RAM-based format (version 2.1), and has appeared variously from Personal Micro Computers, The Peripheral People, and Computer Accessory Technology. It is still available from C.A.T., and (together with the Memory Sidecar) from MSB Electronics, Drawer 766, Barre, Vermont 05641. However, since it contains many of the software drivers and other routines presented in The Custom TRS-80, and represents a complete implementation of the custom interpreter (Chapter 3), it is offered here as a completely revised version in source code format. Its current origin is 3000H, for use with the Memory Sidecar project presented in Chapter 8.

## 74LS00



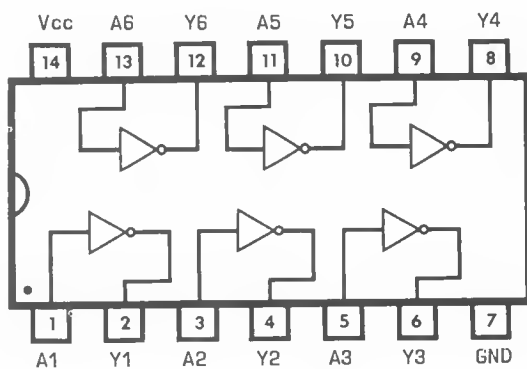
$$Y = \overline{AB}$$

## 74LS02



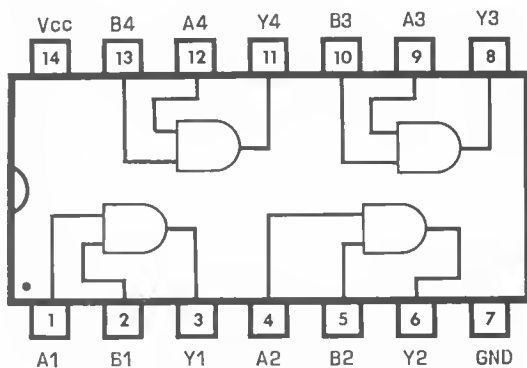
$$Y = \overline{A+B}$$

## 74LS04

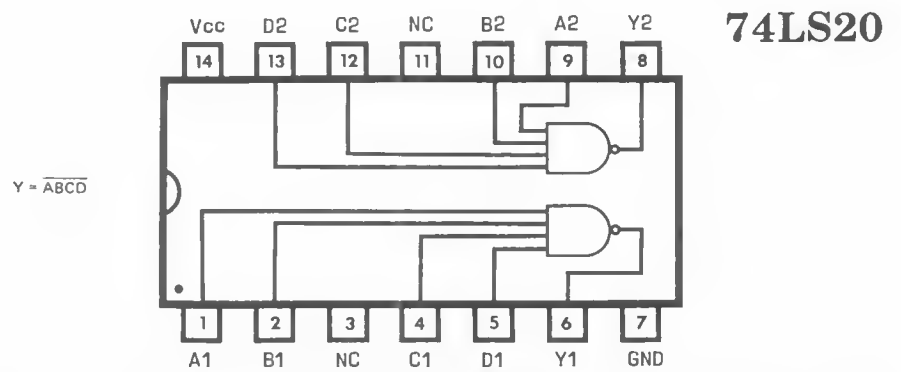
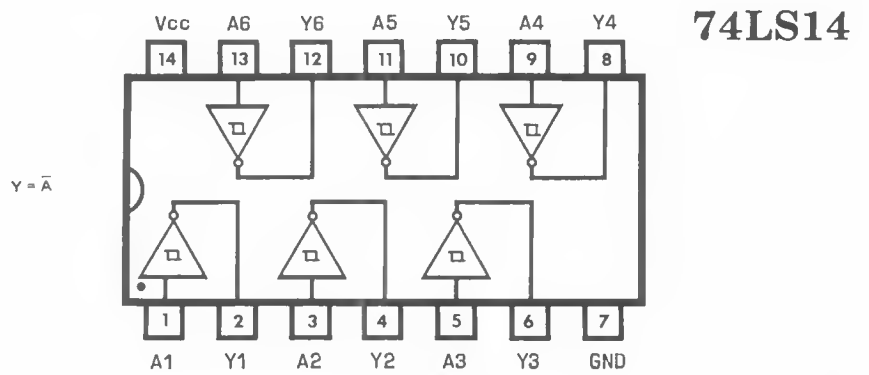
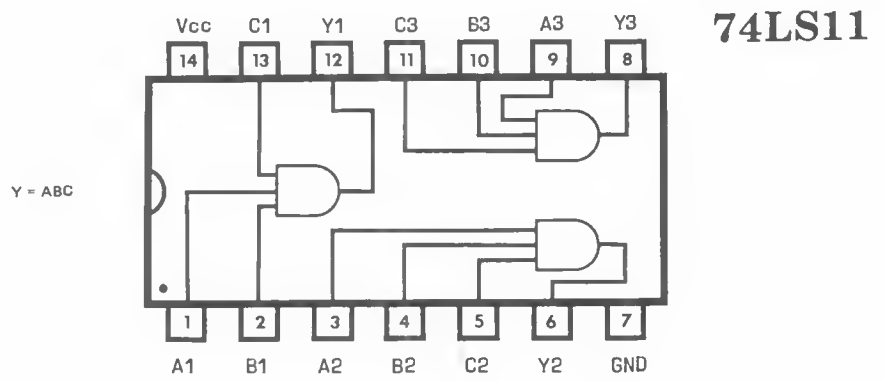
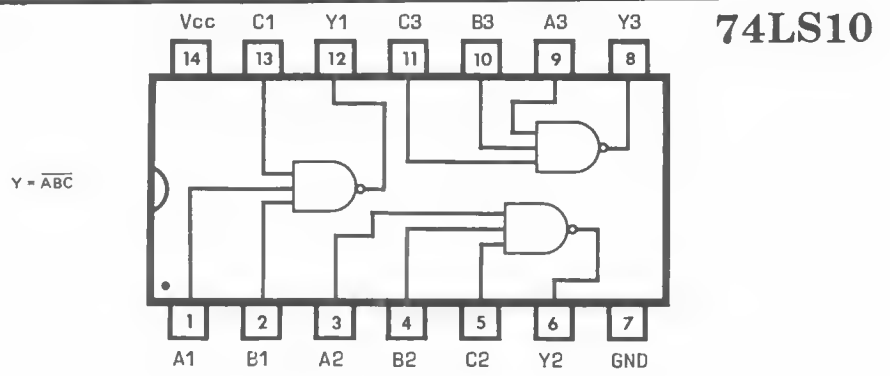


$$Y = \overline{A}$$

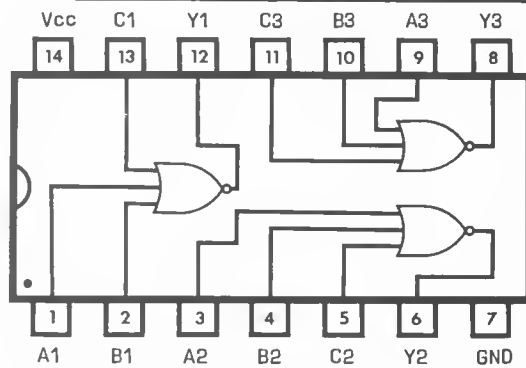
## 74LS08



$$Y = AB$$

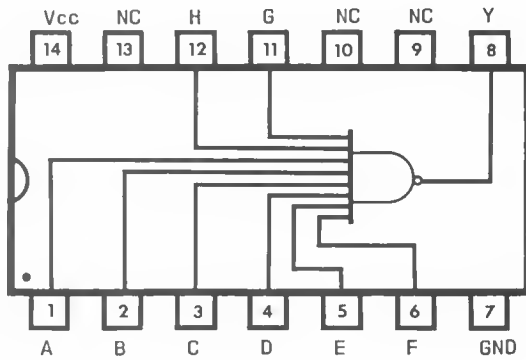


## 74LS27



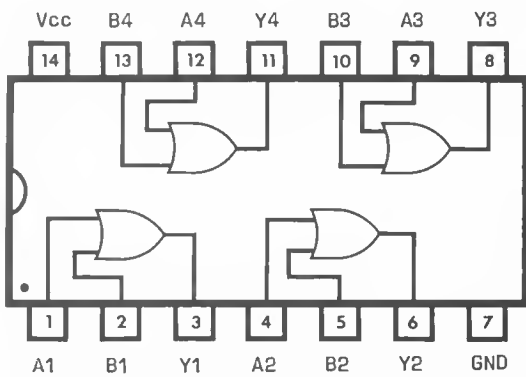
$$Y = A + B + C$$

## 74LS30



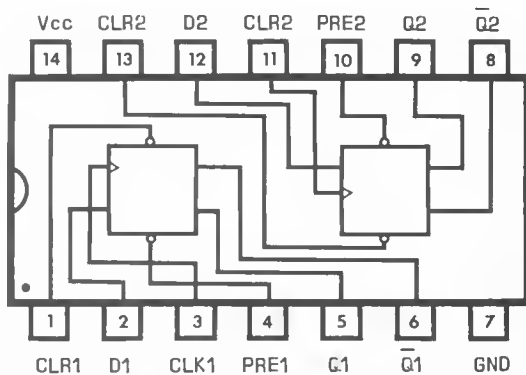
$$Y = ABCDEFGH$$

## 74LS32



$$Y = A + B$$

## 74LS74

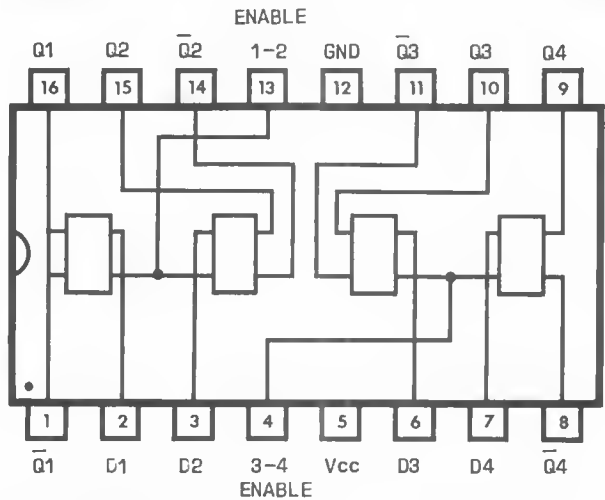


INPUTS				OUTPUTS	
PR	CLR	CLK	D	Q	$\bar{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H*	H*
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q0	$\bar{Q}0$

74LS75

INPUTS		OUTPUTS	
D	G	O	$\bar{O}$
L	H	L	H
H	H	H	L
X	L	$\bar{Q}_0$	$\bar{Q}_0$

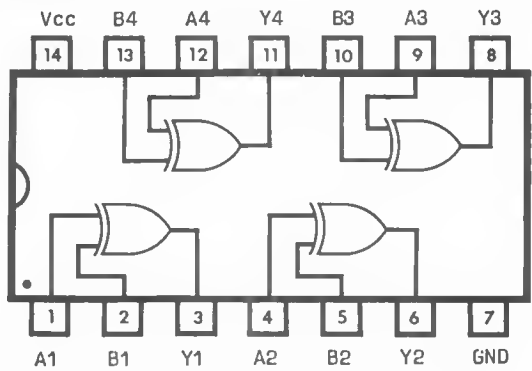
H = High Level, L = Low Level, X = Don't Care  
 $Q_0$  = The Level of Q Before the High-to-Low Transition of G



74LS86

INPUTS		OUTPUT
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

$$Y = A \oplus B = \bar{A}B + A\bar{B}$$



LS90  
BCD COUNT SEQUENCE  
(See Note A)

LS90  
RESET/COUNT TRUTH TABLE

COUNT	OUTPUT			
	$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

LS90 BI-QUINARY (5-2)  
(See Note B)

COUNT	OUTPUT			
	$Q_A$	$Q_D$	$Q_C$	$Q_B$
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	H	L	L	L
6	H	L	L	H
7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

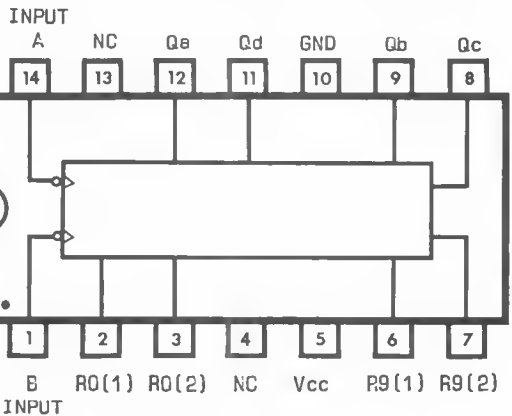
LS92  
RESET/COUNT TRUTH TABLE

RESET INPUTS		OUTPUT			
R0(1)	R0(2)	$Q_D$	$Q_C$	$Q_B$	$Q_A$
H	H	L	L	L	L
L	X	COUNT			
X	L	COUNT			

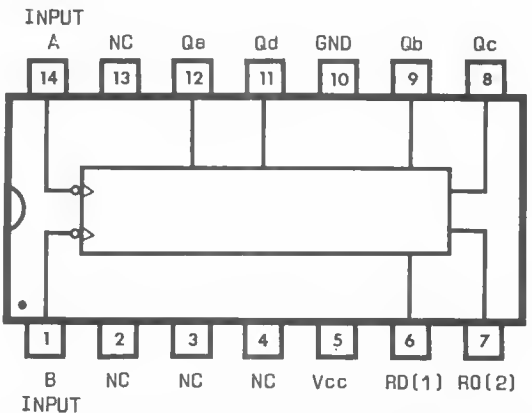
LS92  
COUNT SEQUENCE  
(See Note C)

COUNT	OUTPUT			
	$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	H	L	L	L
7	H	L	L	H
8	H	L	H	L
9	H	L	H	H
10	H	H	L	L
11	H	H	L	H

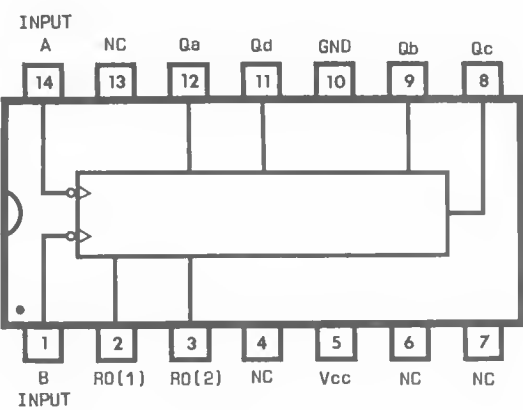
74LS90



74LS92



74LS93



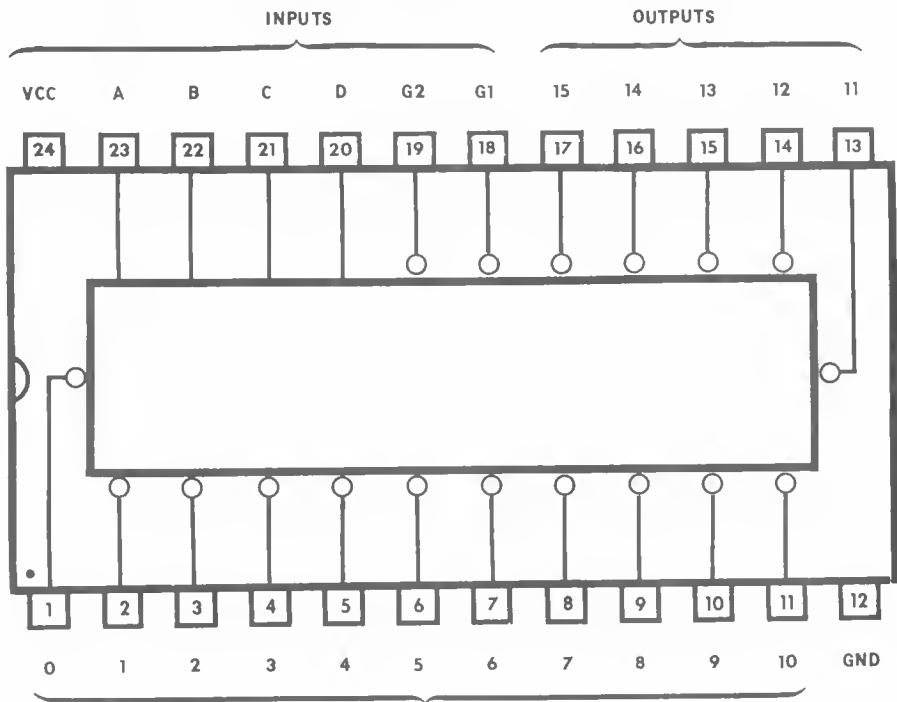
COUNT SEQUENCE  
(See Note C)

COUNT	OUTPUT			
	Q <sub>D</sub>	Q <sub>C</sub>	Q <sub>B</sub>	Q <sub>A</sub>
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

LS93  
Notes:  
(A) Output Q<sub>A</sub> is connected to input B for BCD count.  
(B) Output Q<sub>D</sub> is connected to input A for bi-quinary count.  
(C) Output Q<sub>A</sub> is connected to input B  
(D) H = High Level, L = Low Level, X = Don't Care.

74LS125

74LS154

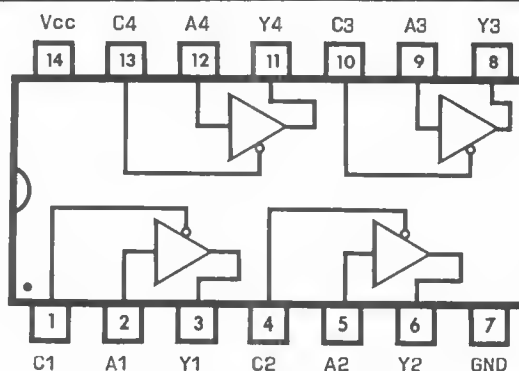


INPUTS						OUTPUTS															
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

H = High Level, L = Low Level, X = Don't Care

INPUTS		OUTPUT
A	C	Y
H	L	H
L	L	L
X	H	H, Z

Y = A

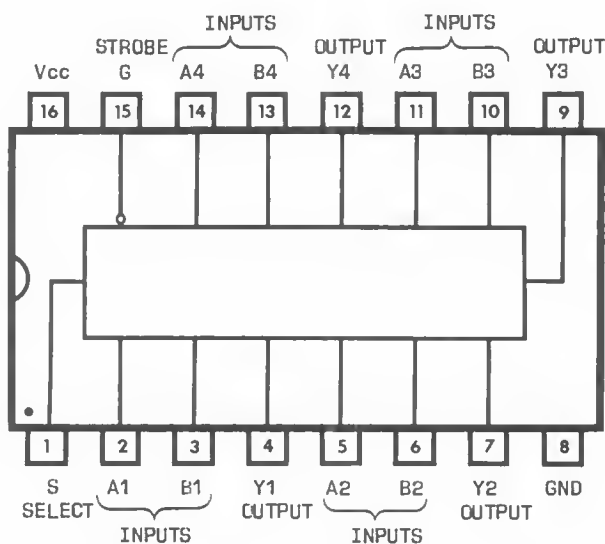


74LS125

74LS154

INPUTS				OUTPUT Y	
STROBE	SELECT	A	B	157, L157A LS157, S157	LS158 S158
H	X	X	X	L	H
L	L	L	X	L	H
L	L	H	X	H	L
L	H	X	L	L	H
L	H	X	H	H	L

H = High Level, L = Low Level, X = Don't Care



74LS157

INPUTS						INTERNAL OUTPUTS		OUTPUT
CLEAR	SHIFT/ LOAD	CLOCK INHIBIT	CLOCK	SERIAL	PARALLEL A...H	Q <sub>A</sub>	Q <sub>B</sub>	Q <sub>H</sub>
L	X	X	X	X	X	L	L	L
H	X	L	L	X	X	Q <sub>A0</sub>	Q <sub>B0</sub>	Q <sub>H0</sub>
H	L	L	↑	X	a...h	a	b	h
H	H	L	↑	H	X	H	Q <sub>An</sub>	Q <sub>Gn</sub>
H	H	L	↑	L	X	L	Q <sub>An</sub>	Q <sub>Gn</sub>
H	X	H	↑	X	X	Q <sub>A0</sub>	Q <sub>B0</sub>	Q <sub>H0</sub>

H = High Level (steady state), L = Low Level (steady state)

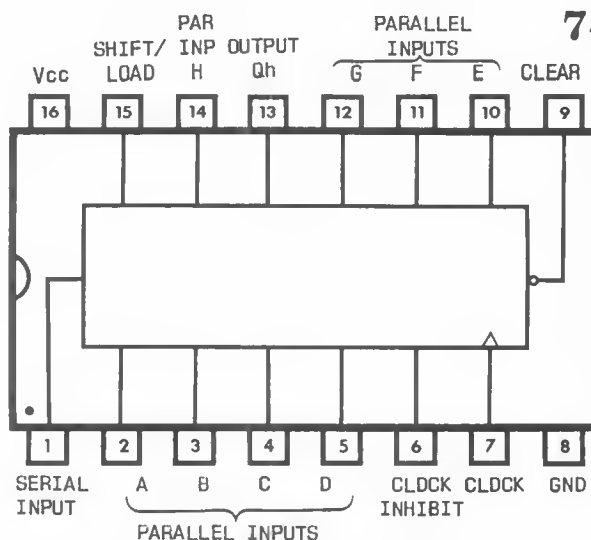
X = Don't Care (any input, including transitions)

↑ = Transition from low to high level

a...h = The level of steady-state input at inputs A through H, respectively.

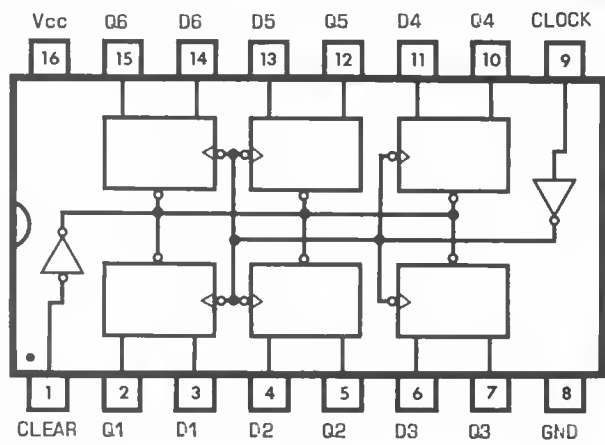
Q<sub>A0</sub>, Q<sub>B0</sub>, Q<sub>H0</sub> = The level of Q<sub>A</sub>, Q<sub>B</sub> or Q<sub>H</sub>, respectively, before the indicated steady-state input conditions were established.

Q<sub>An</sub>, Q<sub>Gn</sub> = The level of Q<sub>A</sub> or Q<sub>G</sub>, respectively, before the most recent ↑ transition of the clock



74LS166

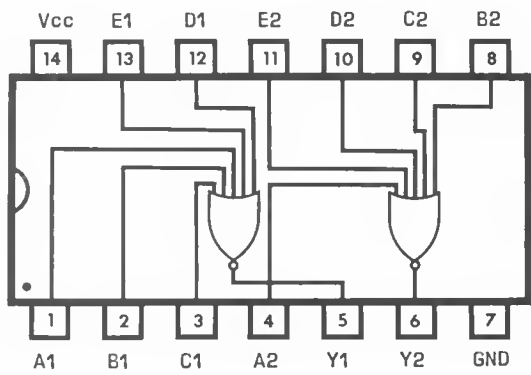
74LS174



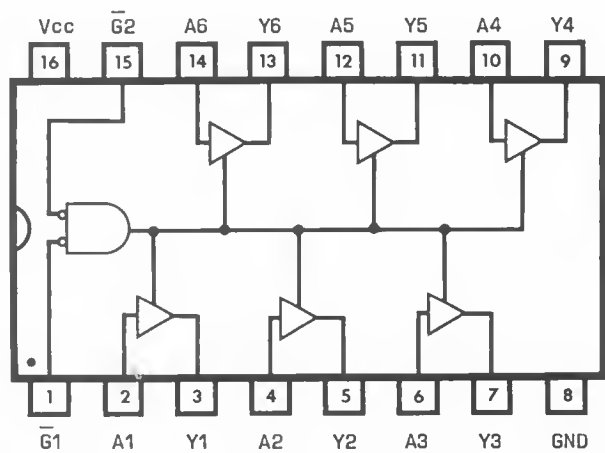
INPUTS			OUTPUTS	
CLEAR	CLOCK	D	Q	$\bar{Q}$
L	X	X	L	H
H	↑	H	H	L
H	↑	L	L	H
H	L	X	$Q_0$	$\bar{Q}_0$

H = High Level (steady state)  
L = Low Level (steady state)  
X = Don't Care  
↑ = Transition from low to high level  
 $Q_0$  = The level of Q before the indicated steady-state input conditions were established.  
↑ = 175, LS175, and S175 only

74LS260



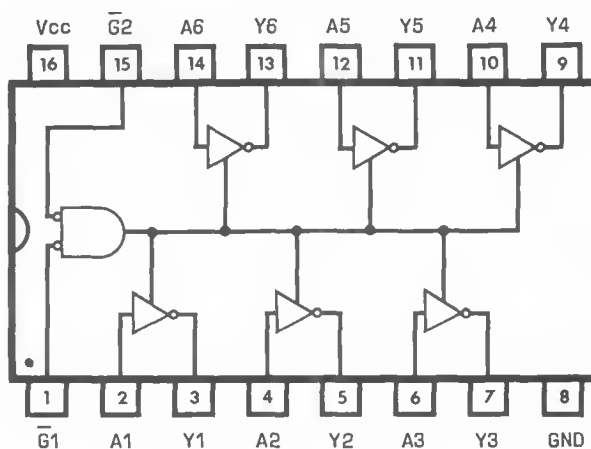
$Y = \overline{A+B+C+D+E}$



INPUTS			OUTPUT
$\bar{G}1$	$\bar{G}2$	A	Y
H	X	X	Z
X	H	X	Z
L	L	H	H
L	L	L	L

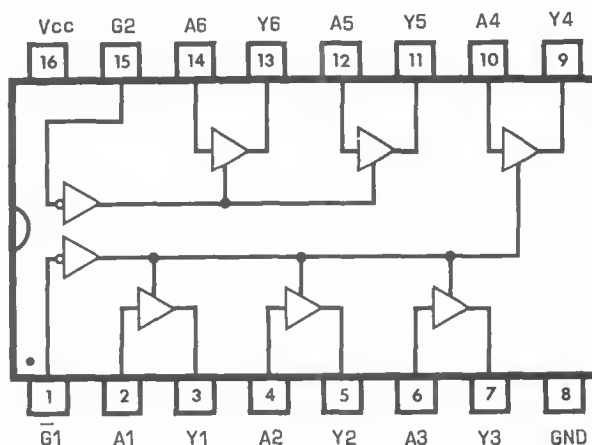
# 74LS366

INPUTS			OUTPUT
$\bar{G}1$	$\bar{G}2$	A	Y
H	X	X	Z
X	H	X	Z
L	L	H	L
L	L	L	H



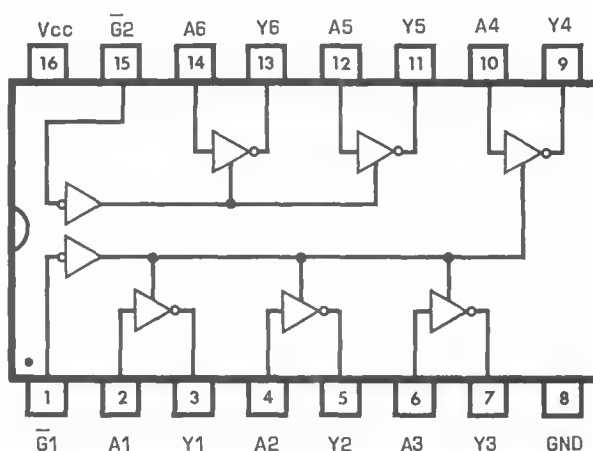
# 74LS367

INPUTS		OUTPUT
$\bar{G}$	A	Y
H	X	Z
L	H	H
L	L	L

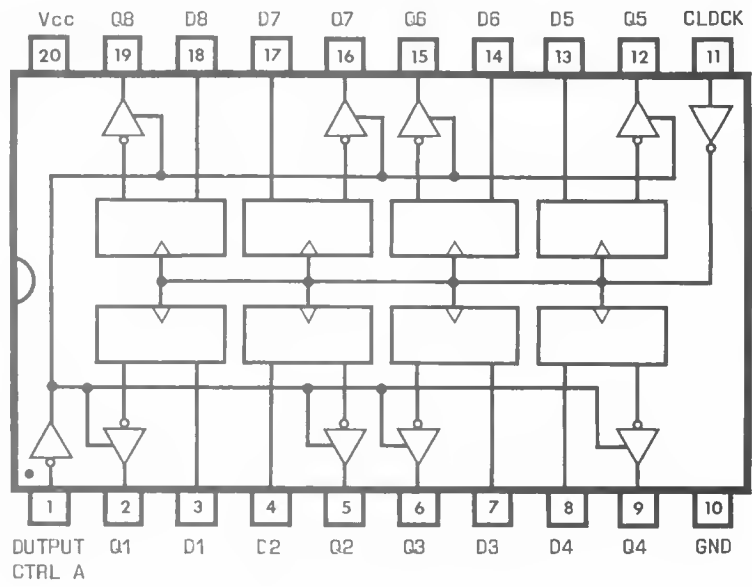


# 74LS368

INPUTS		OUTPUT
$\bar{G}$	A	Y
H	X	Z
L	H	L
L	L	H

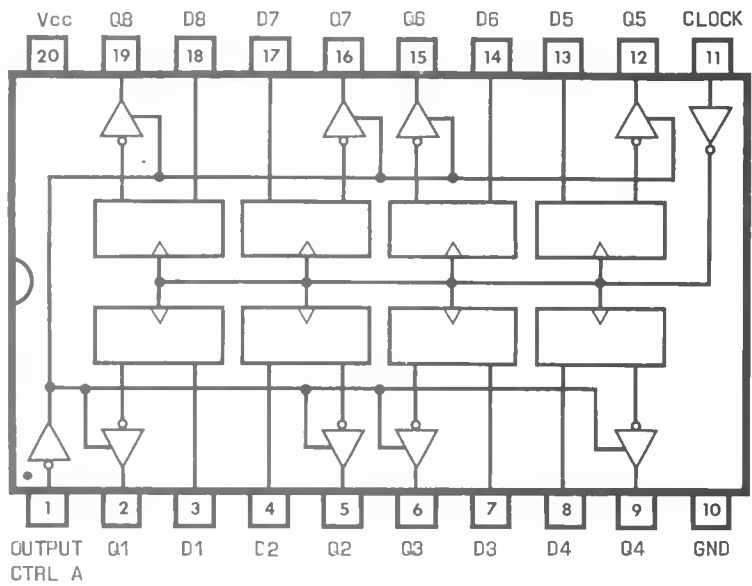


74LS373



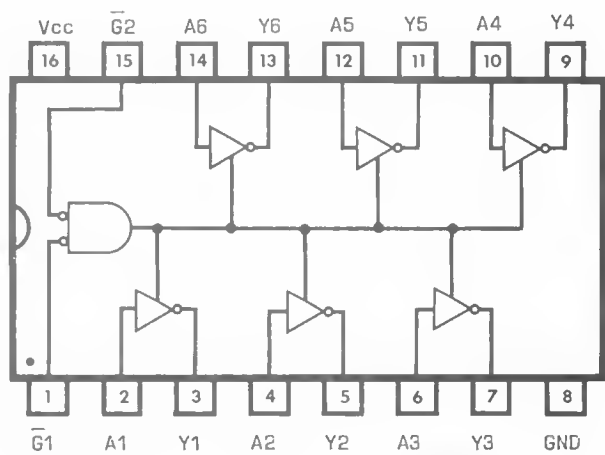
OUTPUT CONTROL	CLOCK	O	OUTPUT
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

74LS374



OUTPUT CONTROL	CLOCK	O	OUTPUT
L	↑	H	H
L	↑	L	L
L	L	X	Q0
H	X	X	Z

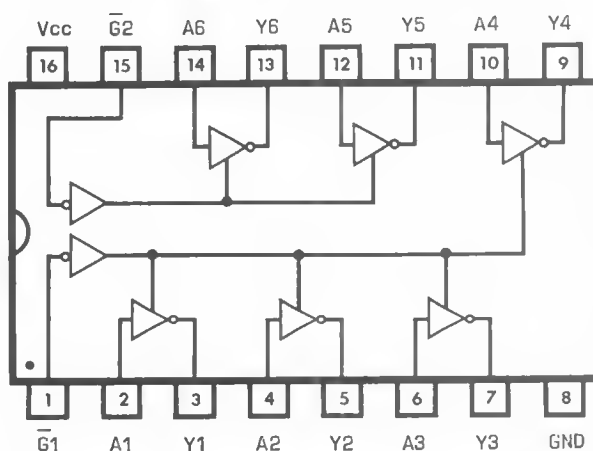
80C96



INPUTS			OUTPUT
G1	G2	A	Y
H	X	X	Hi-Z
X	H	X	Hi-Z
L	L	H	L
L	L	L	H

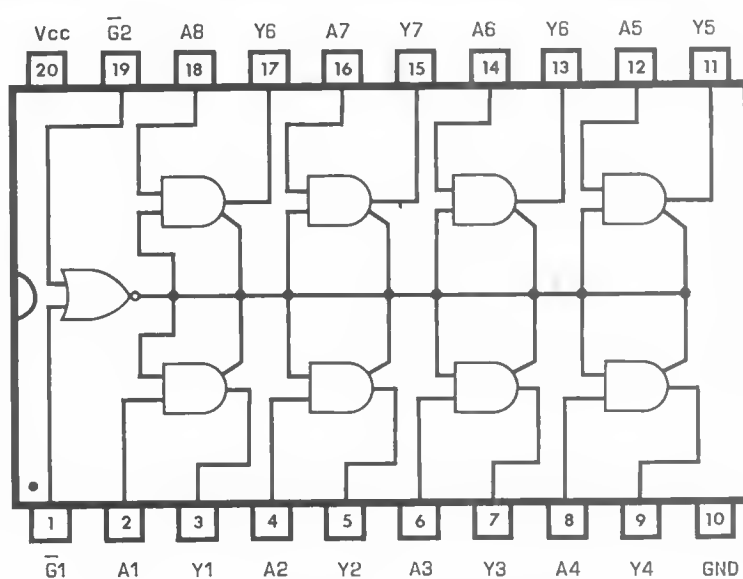
# 80C98

INPUTS		OUTPUT
$\bar{G}$	A	Y
H	X	H <sub>i</sub> -Z
L	H	L
L	L	H



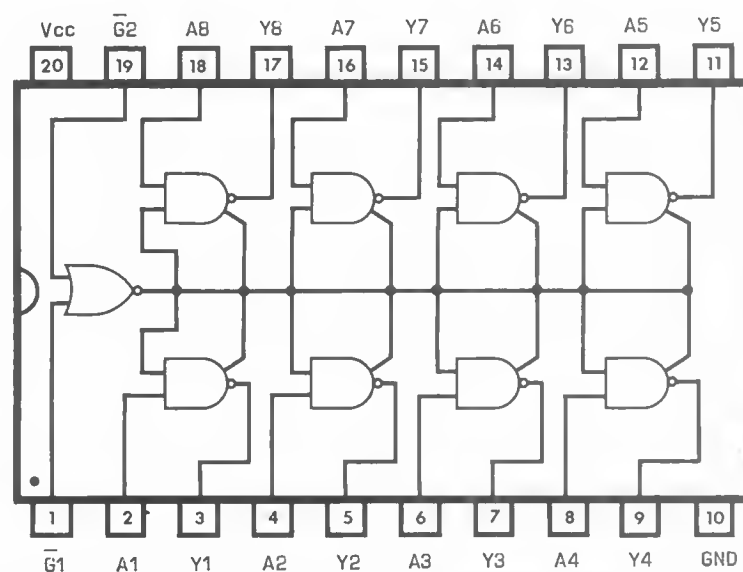
# 81LS95

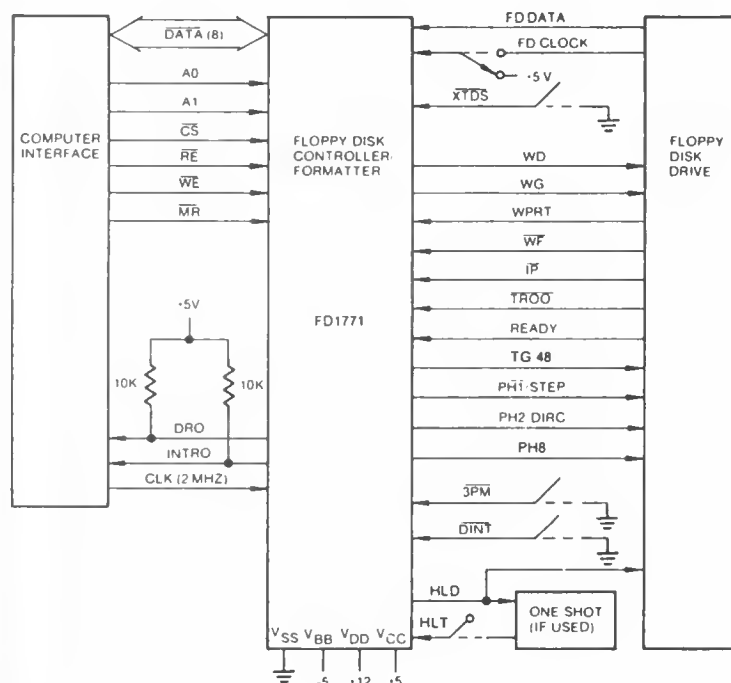
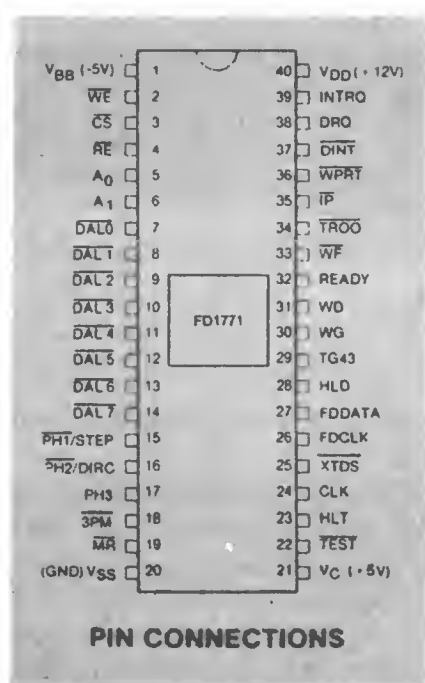
INPUTS			OUTPUT
$\bar{G}1$	$\bar{G}2$	A	Y
H	X	X	Z
X	H	X	Z
L	L	H	H
L	L	L	L



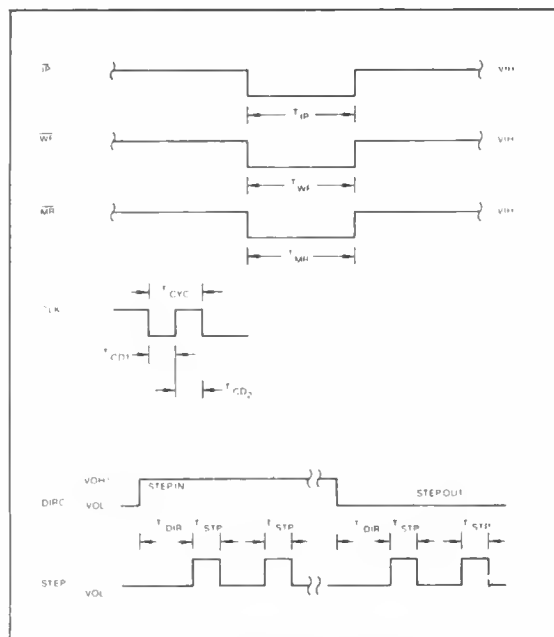
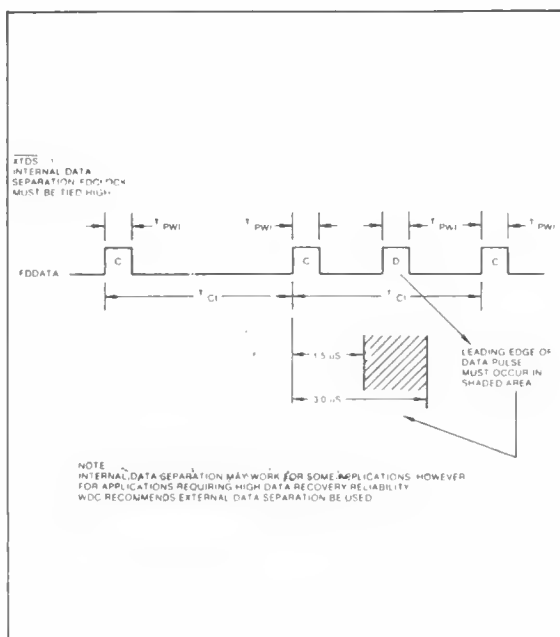
# 81LS96

INPUTS			OUTPUT
$\bar{G}1$	$\bar{G}2$	A	Y
H	X	X	Z
X	H	X	Z
L	L	H	L
L	L	L	H





FD1771 SYSTEM BLOCK DIAGRAM



## PIN OUTS

Pin No.	Pin Name	Symbol	Function																				
1 19	Power Supplies MASTER RESET	$V_{BB}/NC$ $\overline{MR}$	-5V A logic low on this input resets the device and loads "03" into the command register. The Not Ready (Status bit 7) is reset during $\overline{MR}$ ACTIVE. When $\overline{MR}$ is brought to a logic high, a Restore Command is executed, regardless of the state of the Ready signal from the drive.																				
20		$V_{SS}$	Ground																				
21		$V_{CC}$	+5V																				
40		$V_{DD}$	+12V																				
<b>Computer Interface</b>																							
2	$\overline{WRITE\ ENABLE}$	$\overline{WE}$	A logic low on this input gates data on the DAL into the selected register when $\overline{CS}$ is low.																				
3	$\overline{CHIP\ SELECT}$	$\overline{CS}$	A logic low on this input selects the chip and enables computer communication with the device.																				
4	$\overline{READ\ ENABLE}$	$\overline{RE}$	A logic low on this input controls the placement of data from a selected register on the DAL when $\overline{CS}$ is low.																				
5, 6	REGISTER SELECT LINES	$A_0, A_1$	These inputs select the register to receive/transfer data on the DAL lines under $\overline{RE}$ and $\overline{WE}$ control: <table> <tr> <th><math>A_1</math></th><th><math>A_0</math></th><th><math>\overline{RE}</math></th><th><math>\overline{WE}</math></th></tr> <tr> <td>0</td><td>0</td><td>Status Register</td><td>Command Register</td></tr> <tr> <td>0</td><td>1</td><td>Track Register</td><td>Track Register</td></tr> <tr> <td>1</td><td>0</td><td>Sector Register</td><td>Sector Register</td></tr> <tr> <td>1</td><td>1</td><td>Data Register</td><td>Data Register</td></tr> </table>	$A_1$	$A_0$	$\overline{RE}$	$\overline{WE}$	0	0	Status Register	Command Register	0	1	Track Register	Track Register	1	0	Sector Register	Sector Register	1	1	Data Register	Data Register
$A_1$	$A_0$	$\overline{RE}$	$\overline{WE}$																				
0	0	Status Register	Command Register																				
0	1	Track Register	Track Register																				
1	0	Sector Register	Sector Register																				
1	1	Data Register	Data Register																				
7-14	$\overline{DATA\ ACCESS\ LINES}$	$\overline{DAL0-DAL7}$	Eight bit inverted bidirectional bus used for transfer of data, control, and status. This bus is a receiver enabled by $\overline{WE}$ or a transmitter enabled by $\overline{RE}$ .																				
24	CLOCK	CLK	This input requires a free-running 2 MHz $\pm$ 1% square wave clock for internal timing reference.																				
38	DATA REQUEST	DRQ	This open drain output indicates that the DR contains assembled data in Read operations, or the DR is empty in Write operations. This signal is reset when serviced by the computer through reading or loading the DR in Read or Write operation, respectively. Use 10K pull-up resistor to +5.																				
39	INTERRUPT REQUEST	INTRQ	This open drain output is set at the completion or termination of any operation and is reset when a new command is loaded into the command register. Use 10K pull-up resistor to +5.																				
<b>Floppy Disk Interface:</b>																							
15	Phase 1/Step	$\overline{PH1}/STEP$	If the $\overline{3PM}$ input is a logic low the three-phase motor control is selected and $\overline{PH1}$ , $\overline{PH2}$ , and $\overline{PH3}$ outputs																				

Pin No.	Pin Name	Symbol	Function
16	Phase 2/Direction	$\overline{\text{PH2/DIRC}}$	form a one active low signal out of three. $\overline{\text{PH1}}$ is active low after MR. If the 3PM input is a logic high the step and direction motor control is selected. The step output contains a 4 usec high signal for each step and the direction output is active high when stepping in; active low when stepping out.
17	Phase 3	PH3	
18	3-Phase Motor Select	$\overline{\text{3PM}}$	
22	$\overline{\text{TEST}}$	$\overline{\text{TEST}}$	This input is used for testing purposes only and should be tied to +5V or left open by the user.
23	HEAD LOAD TIMING	HLT	The HLT input is sampled after 10 ms. When a logic high is sampled on the HLT input the head is assumed to be engaged.
25	$\overline{\text{EXTERNAL DATA SEPARATION}}$	$\overline{\text{XTDS}}$	A logic low on this input selects external data separation. A logic high or open selects the internal data separator.
26	FLOPPY DISK CLOCK (External Separation)	FDCLOCK	This input receives the externally separated clock when $\overline{\text{XTDS}} = 0$ . If $\overline{\text{XTDS}} = 1$ , this input should be tied to a logic high.
27	FLOPPY DISK DATA	FDDATA	This input receives the raw read disk data if $\overline{\text{XTDS}} = 1$ , or the externally separated data if $\overline{\text{XTDS}} = 0$ .
28	HEAD LOAD	HLD	The HLD output controls the loading of the Read-Write head against the media.
29	Track Greater than 43	TG43	This output informs the drive that the Read-Write head is positioned between tracks 44-76. This output is valid only during Read and Write commands.
30	WRITE GATE	WG	This output is made valid when writing is to be performed on the diskette.
31	WRITE DATA	WD	This output contains both clock and data bits of 500 ns duration.
32	Ready	READY	This input indicates disk readiness and is sampled for a logic high before Read or Write commands are performed. If Ready is low, the Read or Write operation is not performed and an interrupt is generated. A Seek operation is performed regardless of the state of Ready. The Ready input appears in inverted format as Status Register bit 7.
33	$\overline{\text{WRITE FAULT}}$	$\overline{\text{WF}}$	This input detects wiring faults indications from the drive. When WG=1 and $\overline{\text{WF}}$ goes low, the current Write command is terminated and the Write Fault status bit is set. The $\overline{\text{WF}}$ input should be made inactive (high) when WG becomes inactive.
34	$\overline{\text{TRACK 00}}$	$\overline{\text{TR00}}$	This input informs the FD1771 that the Read-Write head is positioned over Track 00 when a logic low.
35	$\overline{\text{INDEX PULSE}}$	$\overline{\text{IP}}$	Input, when low for a minimum of 10 usec, informs the FD1771 when an index mark is encountered on the diskette.
36	$\overline{\text{WRITE PROTECT}}$	$\overline{\text{WPRT}}$	This input is sampled whenever a Write command is received. A logic low terminates the command and sets the Write Protect status bit.
37	$\overline{\text{DISK INITIALIZATION}}$	$\overline{\text{DINT}}$	The input is sampled whenever a Write Track command is received. If $\overline{\text{DINT}} = 0$ , the operation is terminated and the Write Protect status bit is set.

## COMMAND DESCRIPTION

The FD1771 will accept and execute eleven commands. Command words should only be loaded in the Command Register when the Busy status bit is off (status bit 0). The one exception is the Force Interrupt command. Whenever a command is being executed, the Busy status bit is set. When a command is completed, an interrupt is generated and the Busy status bit is reset. The Status Register indicates whether the completed command encountered an error or was fault-free. For ease of discussion, commands are divided into four types. Commands and types are summarized in Table 2.

### TYPE 1 COMMANDS

The Type 1 Commands include the RESTORE, SEEK, STEP, STEP-IN, and STEP-OUT commands. Each of the Type 1 Commands contain a rate field ( $r_0r_1$ ), which determines the stepping motor rate as defined in Table 1, page 4.

The Type 1 Commands contain a head load flag (h) which determines if the head is to be loaded at the

Table 2. COMMAND SUMMARY

TYPE	COMMAND	BITS							
		7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	$r_1$	$r_0$
I	Seek	0	0	0	1	h	V	$r_1$	$r_0$
I	Step	0	0	1	u	h	V	$r_1$	$r_0$
I	Step In	0	1	0	u	h	V	$r_1$	$r_0$
I	Step Out	0	1	1	u	h	V	$r_1$	$r_0$
II	Read Command	1	0	0	0	1	1	0	0
II	Write Command	1	0	1	0	1	1	0	$a_0$
III	Read Address	1	1	0	0	0	E	0	0
III	Read Track	1	1	1	0	0	1	0	$\bar{s}$
III	Write Track	1	1	1	1	0	1	0	0
IV	Force Interrupt	1	1	0	1	$l_3$	$l_2$	$l_1$	$l_0$

Note: Bits shown in TRUE form.

Table 3. FLAG SUMMARY

TYPE I
<h>h = Head Load flag (Bit 3)</h> <p>h = 1, Load head at beginning h = 0, Do not load head at beginning</p> <h>V = Verify flag (Bit 2)</h> <p>V = 1, Verify on last track V = 0, No verify</p> <h><math>r_1r_0</math> = Stepping motor rate (Bits 1-0)</h> <p>Refer to Table 1 for rate summary</p> <h>u = Update flag (Bit 4)</h> <p>u = 1, Update Track register u = 0, No update</p>

Table 4. FLAG SUMMARY

TYPE II
<h>m = Multiple Record flag (Bit 4)</h> <p>m=0, Single Record m=1, Multiple Records</p> <h>b = Block length flag (Bit 3)</h> <p>b=1, IBM format (128 to 1024 bytes) b=0, Non-IBM format (16 to 4096 bytes)</p> <h><math>a_1a_0</math> = Data Address Mark (Bits 1-0)</h> <p><math>a_1a_0</math> = 00, FB (Data Mark) <math>a_1a_0</math> = 01, FA (User defined) <math>a_1a_0</math> = 10, F9 (User defined) <math>a_1a_0</math> = 11, F8 (Deleted Data Mark)</p>

Table 5. FLAG SUMMARY

TYPE III
<h>s = Synchronize flag (Bit 0)</h> <p><math>\bar{s}</math>=0, Synchronize to AM <math>\bar{s}</math>=1, Do Not Synchronize to AM</p>
TYPE IV
<h><math>l_i</math> = Interrupt Condition flags (Bits 3-0)</h> <p><math>l_0</math>=1, Not Ready to Ready Transition <math>l_1</math>=1, Ready to Not Ready Transition <math>l_2</math>=1, Index Pulse <math>l_3</math>=1, Immediate interrupt</p> <h>E = Enable HLD and 10 msec Delay</h> <p>E=1, Enable HLD, HLT and 10 msec Delay E=0, Head is assumed Engaged and there is no 10 msec Delay</p>

beginning of the command. If h=1, the head is loaded at the beginning of the command (HLD output is made active). If h=0, HLD is deactivated. Once the head is loaded, the head will remain engaged until the FD1771 receives a command that specifically disengages the head. If the FD1771 does not receive any commands after two revolutions of the disk, the head will be automatically disengaged (HLD made inactive). The Head Load Timing Input is sampled after a 10 ms delay, when reading or writing on the disk is to occur.

The Type 1 Commands also contain a verification (V) flag which determines if a verification operation is to take place on the destination track. If V=1, a verification is performed; if V=0, no verification is performed.

During verification, the head is loaded and after an internal 10 ms delay, the HLT input is sampled. When

## ELECTRICAL CHARACTERISTICS

### Maximum Ratings

V <sub>DD</sub> with respect to V <sub>BB</sub> (Ground)	+20 to -0.3V
Max Voltage to any input with respect to V <sub>BB</sub>	+20 to -0.3V
Operating Temperature	0°C to 70°C
Storage Temperature	-55°C to +125°C

## OPERATING CHARACTERISTICS (DC)

T <sub>A</sub> = 0°C to 70°C, V <sub>DD</sub> = +12.0V ± .6V, V <sub>BB</sub> = -5.0 ± .5V, V <sub>SS</sub> = 0V, V <sub>CC</sub> = +5V ± .25V I <sub>DD</sub> = 10 ma Nominal, I <sub>CC</sub> = 30 ma Nominal, I <sub>BB</sub> = 0.4 µa Nominal
--

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
I <sub>LI</sub>	Input Leakage			10	µA	V <sub>IN</sub> = V <sub>DD</sub>
I <sub>LO</sub>	Output Leakage			10	µA	V <sub>OUT</sub> = V <sub>DD</sub>
V <sub>IH</sub>	Input High Voltage	2.6			V	
V <sub>IL</sub>	Input Low Voltage (All Inputs)			0.8	V	
V <sub>OH</sub>	Output High Voltage	2.8			V	I <sub>O</sub> = -100 µA
V <sub>OL</sub>	Output Low Voltage			0.45**	V	I <sub>O</sub> = 1.0 mA

\*\*Write Gate V<sub>OL</sub> ≤ 0.5V.

## TIMING CHARACTERISTICS

T <sub>A</sub> = 0°C to 70°C, V <sub>DD</sub> = +12V ± .6V, V <sub>BB</sub> = -5V ± .25V, V <sub>SS</sub> = 0V, V <sub>CC</sub> = +5V ± .25V
---

NOTE: Timings are given for 2 MHz Clock. For those timings noted, values will double when chip is operated at 1 MHz. Use 1 MHz when using mini-floppy.

### Read Operations

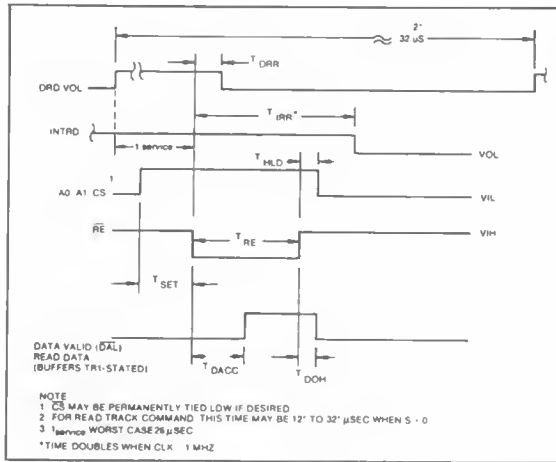
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TSET	Setup ADDR and CS to $\overline{RE}$	100			nsec	C <sub>L</sub> = 25 pf
THLD	Hold ADDR and CS from $\overline{RE}$	10			nsec	
TRE	$\overline{RE}$ Pulse Width	500			nsec	
TDRR	DRQ Reset from $\overline{RE}$			500	nsec	
TIRR	INTRQ Reset from $\overline{RE}$			3000	nsec	C <sub>L</sub> = 25 pf
TDACC	Data Access from $\overline{RE}$			450	nsec	
TDOH	Data Hold from $\overline{RE}$	50		150	nsec	C <sub>L</sub> = 25 pf

### Write Operations

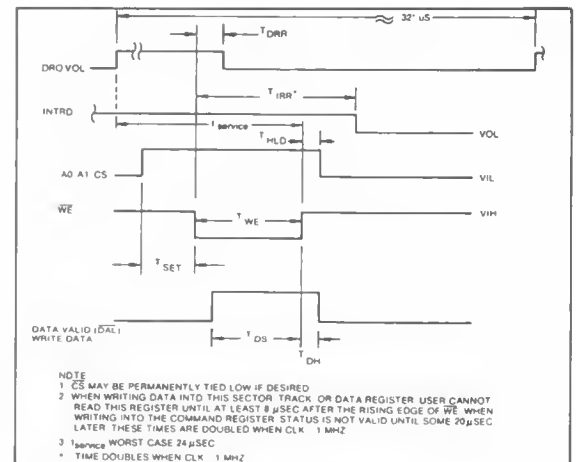
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TSET	Setup ADDR and CS to $\overline{WE}$	100			nsec	See Note
THLD	Hold ADDR and CS from $\overline{WE}$	10			nsec	
TWE	$\overline{WE}$ Pulse Width	350			nsec	
TDRR	DRQ Reset from $\overline{WE}$			500	nsec	
TIRR	INTRQ Reset from $\overline{WE}$			3000	nsec	
TDS	Data Setup to $\overline{WE}$	250			nsec	
TDH	Data Hold from $\overline{WE}$	150			nsec	

### External Data Separation (XTDS = 0)

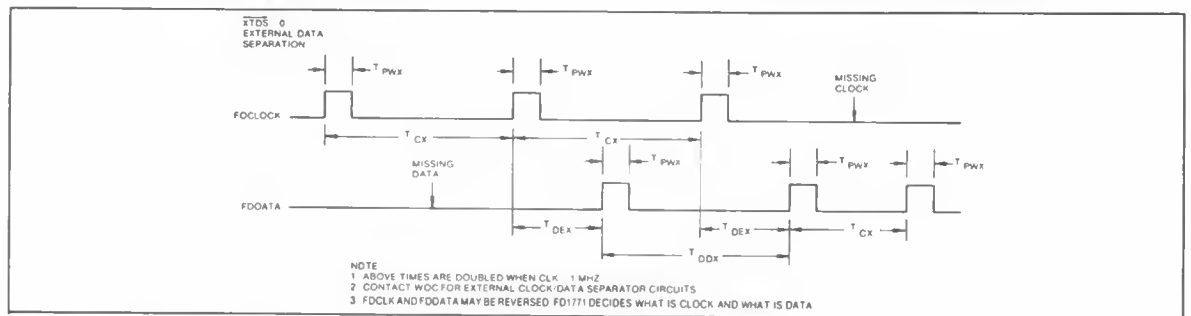
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TPWX	Pulse Width Read Data & Read Clock	150		350	nsec	
TCX	Clock Cycle External	2500			nsec	
TDEX	Data to Clock	500			nsec	
TDDX	Data to Data Cycle	2500			nsec	



READ ENABLE TIMING



WRITE ENABLE TIMING



READ TIMING (XTDS = 0)

Internal Data Separation (XTDS = 1)

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TPWI	Pulse Width Data and Clock	150		1000	nsec	
TCI	Clock Cycle Internal	3500		5000	nsec	

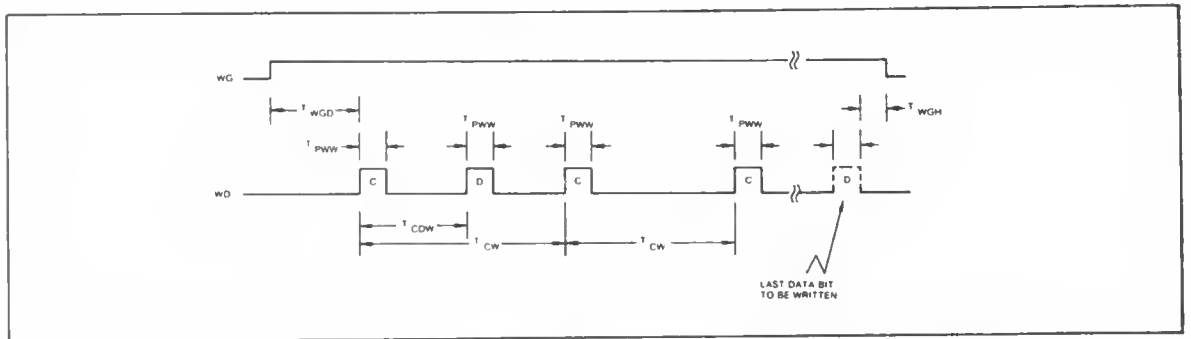
Write Data Timing

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TWGD	Write Gate to Data		1200		nsec	300 nsec ± CLK tolerance
TPWW	Pulse Width Write Data	500		600	nsec	
TCDW	Clock to Data		2000		nsec	± CLK tolerance
TCW	Clock Cycle Write		4000		nsec	± CLK tolerance
TWGH	Write Gate Hold to Data	0		100	nsec	

Miscellaneous Timing

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TCD <sub>1</sub>	Clock Duty	175			nsec	2 MHz ± 1% See Note
TCD <sub>2</sub>	Clock Duty	210			nsec	
TSTP	Step Pulse Output	3800		4200	nsec	
TDIR	Direct Setup to Step	24			usec	
TMR	Master Reset Pulse Width	10			usec	
TIP	Index Pulse Width	10			usec	
TWF	Write Fault Pulse Width	10			usec	

These times doubled when CLK = 1 MHz

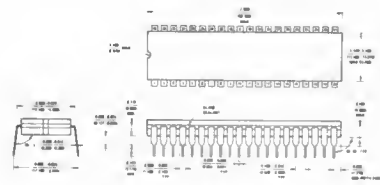


**WRITE DATA TIMING**

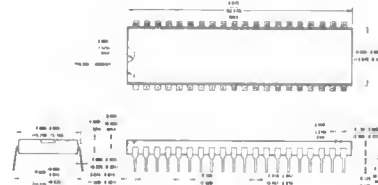
Mode Definition Summary Table

Port Bits	Mode 0		Mode 1		Mode 2
	IN	OUT	IN	OUT	Group A Only
PA <sub>0</sub>	IN	OUT	IN	OUT	Bidirectional ↑ ↓ Bidirectional
PA <sub>1</sub>	IN	OUT	IN	OUT	
PA <sub>2</sub>	IN	OUT	IN	OUT	
PA <sub>3</sub>	IN	OUT	IN	OUT	
PA <sub>4</sub>	IN	OUT	IN	OUT	
PA <sub>5</sub>	IN	OUT	IN	OUT	
PA <sub>6</sub>	IN	OUT	IN	OUT	
PA <sub>7</sub>	IN	OUT	IN	OUT	
PB <sub>0</sub>	IN	OUT	IN	OUT	(Mode 0 or Mode 1 only)
PB <sub>1</sub>	IN	OUT	IN	OUT	
PB <sub>2</sub>	IN	OUT	IN	OUT	
PB <sub>3</sub>	IN	OUT	IN	OUT	
PB <sub>4</sub>	IN	OUT	IN	OUT	
PB <sub>5</sub>	IN	OUT	IN	OUT	
PB <sub>6</sub>	IN	OUT	IN	OUT	
PB <sub>7</sub>	IN	OUT	IN	OUT	
PC <sub>0</sub>	IN	OUT	INTR <sub>B</sub>	INTR <sub>B</sub>	I/O
PC <sub>1</sub>	IN	OUT	IBF <sub>B</sub>	ÖBF <sub>B</sub>	I/O
PC <sub>2</sub>	IN	OUT	STB <sub>B</sub>	ACK <sub>B</sub>	I/O
PC <sub>3</sub>	IN	OUT	INTR <sub>A</sub>	INTR <sub>A</sub>	INTR <sub>A</sub>
PC <sub>4</sub>	IN	OUT	STB <sub>A</sub>	I/O	STB <sub>A</sub>
PC <sub>5</sub>	IN	OUT	IBF <sub>A</sub>	I/O	IBF <sub>A</sub>
PC <sub>6</sub>	IN	OUT	I/O	ACK <sub>A</sub>	ACK <sub>A</sub>
PC <sub>7</sub>	IN	OUT	I/O	ÖBF <sub>A</sub>	ÖBF <sub>A</sub>

Physical Dimensions



Ceramc Dual-in-Line Package [Cerdip (J)]  
Order Number INS8255J



Plastic Dual-in-Line Package (N)  
Order Number INS8255N



National Semiconductor Corporation  
2900 Semiconductor Drive  
Santa Clara, CA 95051  
Tel: (408) 737-5000  
TWA: (910) 339-8240

National Semiconductor GmbH  
Edenheimerstrasse 61-11  
8000 München 21  
West Germany  
Tel: (089) 576091  
Telex: 05-22772

NS International Inc. Japan  
Miyake Building  
1-9 Yokoyuki, Shimizu-ku 180  
Tokyo, Japan  
Tel: (03) 355-3711  
TWA: 232-3015 NSCJ—J

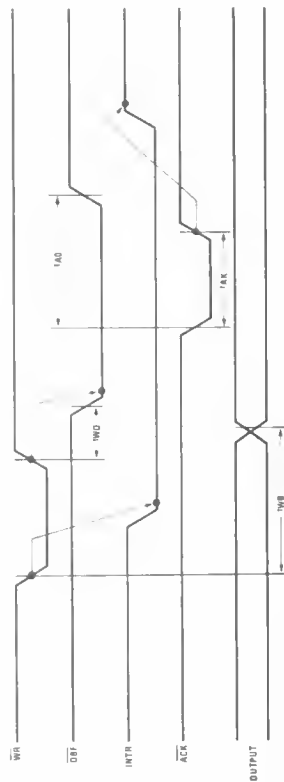
National Semiconductor (Hong Kong) Ltd.  
8th Floor  
Cheung Kong Electronic Bldg  
4 Hung To Street  
Kowloon, Hong Kong  
Tel: 3-899235  
Telex: 43366 NSEHK HK  
Cable: NATSEM

NS Electronics Do Brasil  
Avda. Brigadier Faria Lima 844  
11 Andar, Conjunto 1104  
Jardim Paulistano  
São Paulo, Brasil  
Telex: 1121008 Cabine São Paulo Telex 32096

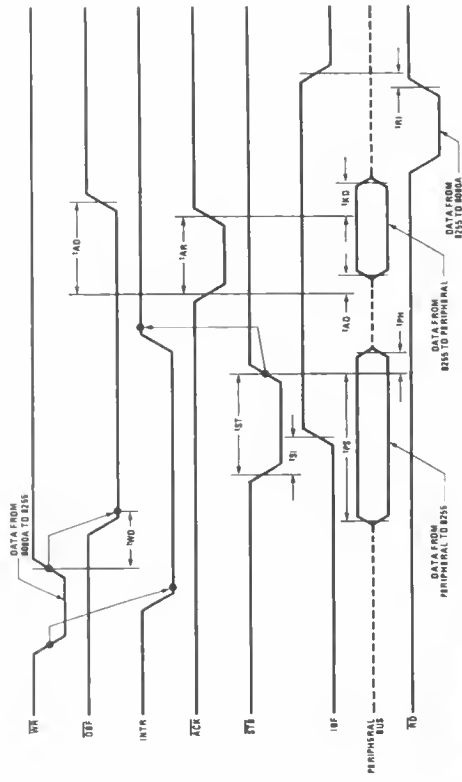
NS Electronics Pty. Ltd.  
Chr. Stud Rd. & Men. Highway  
Bayswater, Victoria 3153  
Australia  
Tel: 03-729-6333

National does not assume any responsibility for use of any circuitry described; no circuit patent licenses are implied; and National reserves the right at any time without notice to change said circuitry.

## Timing Waveforms (cont'd.)



Mode 1 (Strobed Output)



### Mode 2 (Bidirectional)

### Mode 2 (Bidirectional)

## INPUT/OUTPUT SIGNALS

Data (D7-D0) Bus, Pins 27-34: This bus comprises eight TRI-STATE® input/output lines. The bus provides bidirectional communication between the INS8255 and the INS8080A. Data is routed to or from the internal data bus buffer upon execution of an OUT or IN Instruction, respectively, by the INS8080A. In addition, control words and status information are transferred through the data bus buffer.

Port A (PA7-PA0), Pins 37-40: This 8-bit input/output port forms one 8-bit data output latch/buffer and/or one 8-bit data input latch.

## NOTE

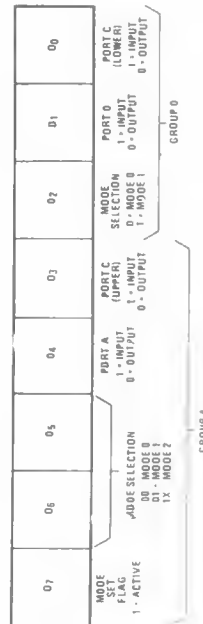
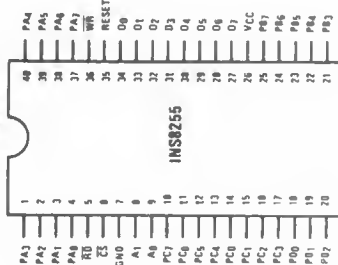
The system software uses a Mode Definition Control Word (see figure) as the second byte of OUT Instruction(s) to program the functional configuration of Ports A through C. Whenever the mode is changed, all output registers (and status flip-flops) are reset.

Port B (PB7-PB0), Pins 18-25: This 8-bit input/output port forms one 8-bit data output latch/buffer or one 8-bit data input buffer.

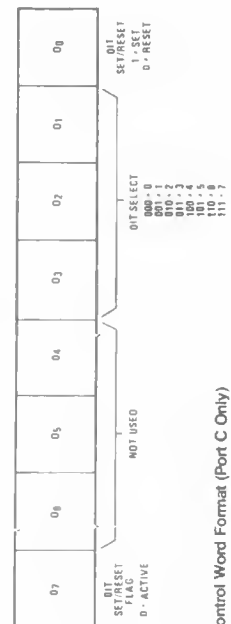
Port C (PC7-PC0), Pins 10-17: This 8-bit input/output port forms one 8-bit data output latch/buffer or one 8-bit data input buffer. The port can be split into two 4-bit ports under the mode control. Each of these 4-bit ports contains a 4-bit latch that may be used for the control and status signals, in conjunction with Ports A

and B. The system software includes a Bit Set/Reset Control Word (see figure) for setting or resetting any of the eight bits of Port C. When Port C is being used as a status/control for Port A or B, the Port C bits can be set or reset by using the Bit Set/Reset Control Word as the second byte of OUT Instruction(s).

## Pin Configuration



Mode Definition Control Word Format



Bit Set/Reset Control Word Format (Port C Only)

## Operating Modes

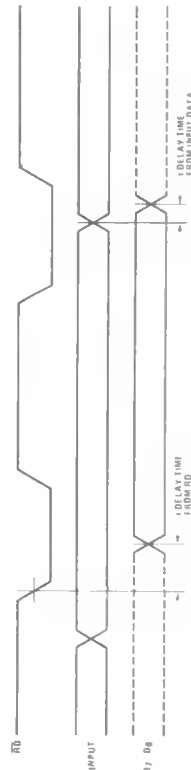
## Mode 0 (Basic Input/Output)

In this mode, simple input and output operations for each of the three ports are provided. No "handshaking" is required; data is simply written to or read from a specified port.

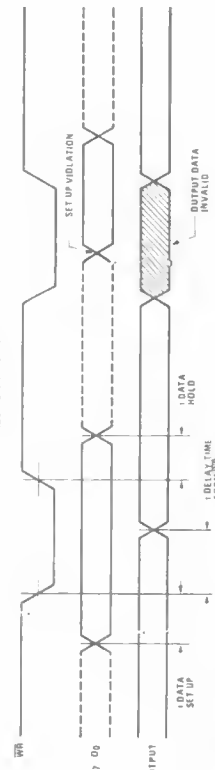
## Mode 0 Port Definition Chart

No.	Control Word Bits								Group A			Group B		
	D7	D6	D5	D4	D3	D2	D1	D0	Port A (Upper)	Port A (Lower)	Port B (Upper)	Port B (Lower)	Port C (Upper)	Port C (Lower)
0	1	0	0	0	0	0	0	0	OUTPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
1	1	0	0	0	0	0	0	1	OUTPUT	OUTPUT	OUTPUT	OUTPUT	INPUT	INPUT
2	1	0	0	0	0	0	1	0	OUTPUT	OUTPUT	OUTPUT	OUTPUT	INPUT	INPUT
3	1	0	0	0	0	0	1	1	OUTPUT	OUTPUT	OUTPUT	OUTPUT	INPUT	INPUT
4	1	0	0	0	0	1	0	0	OUTPUT	OUTPUT	OUTPUT	OUTPUT	INPUT	INPUT
5	1	0	0	0	0	1	0	1	OUTPUT	OUTPUT	OUTPUT	OUTPUT	INPUT	INPUT
6	1	0	0	0	0	1	0	1	OUTPUT	OUTPUT	OUTPUT	OUTPUT	INPUT	INPUT
7	1	0	0	0	0	1	0	1	OUTPUT	OUTPUT	OUTPUT	OUTPUT	INPUT	INPUT
8	1	0	0	0	1	0	0	0	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
9	1	0	0	0	1	0	0	0	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
10	1	0	0	0	1	0	0	1	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
11	1	0	0	0	1	0	0	1	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
12	1	0	0	0	1	0	0	0	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
13	1	0	0	0	1	0	0	1	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
14	1	0	0	0	1	0	1	0	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
15	1	0	0	0	1	0	1	1	INPUT	INPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT

BASIC INPUT TIMING (OUTPUTS LATCHED)



BASIC OUTPUT TIMING (OUTPUTS LATCHED)

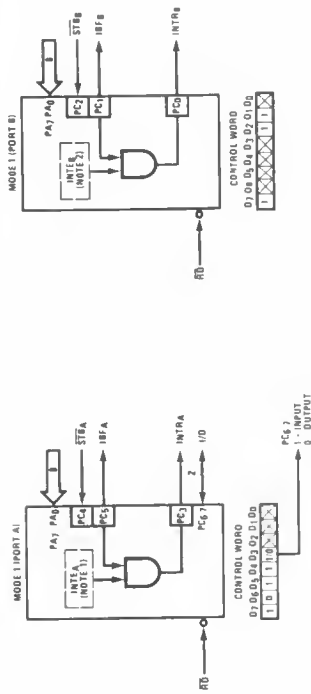


Mode 0 Timing

## Operating Modes (cont'd.)

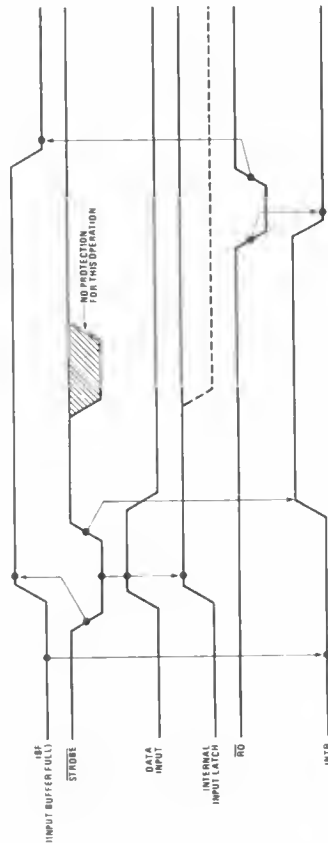
### Mode 1 (Strobed Input/Output)

In this mode, a means for transferring input/output data to or from a specified port in conjunction with strobes or "handshaking" signals is provided. Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals in Mode 1. The programmer can read the contents of Port C to test or verify the status of each peripheral device. Since no special instruction is provided in the INS8080A microcomputer system to read the Port C status information, a normal read operation must be executed to perform this function.



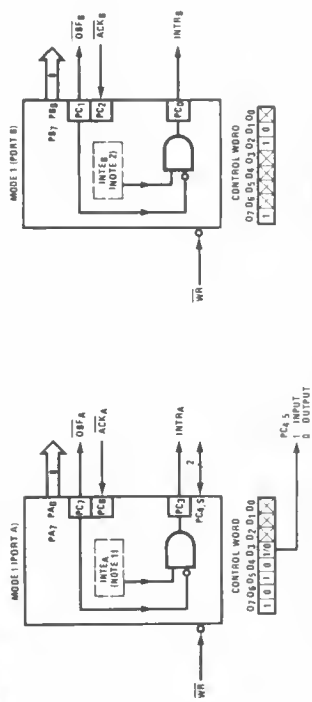
- Notes
1. INTEA is controlled by bit set/reset of PC4
  2. INTEB is controlled by bit set/reset of PC2

### Mode 1 Input



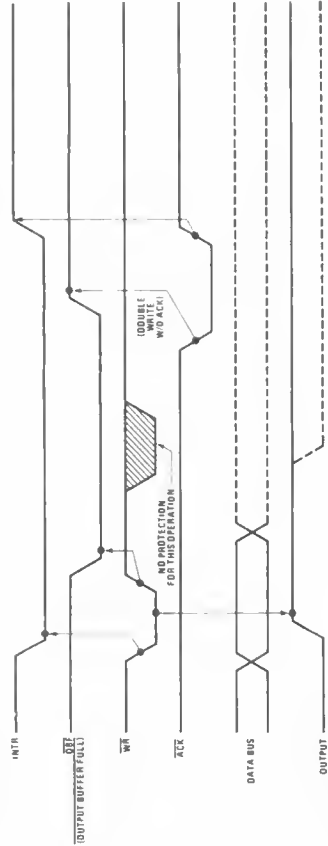
### Mode 1 Input Timing

## Operating Modes (cont'd.)

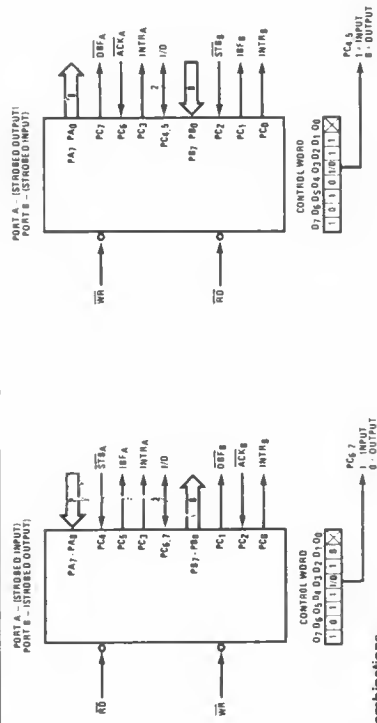


- Notes
1. INTEA is controlled by bit set/reset of PC4
  2. INTEB is controlled by bit set/reset of PC2

### Mode 1 Output



### Mode 1 Output Timing

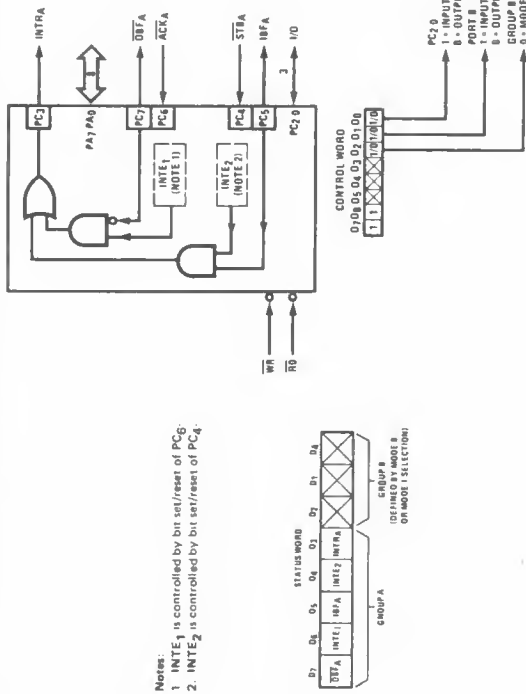


### Mode 1 Combinations

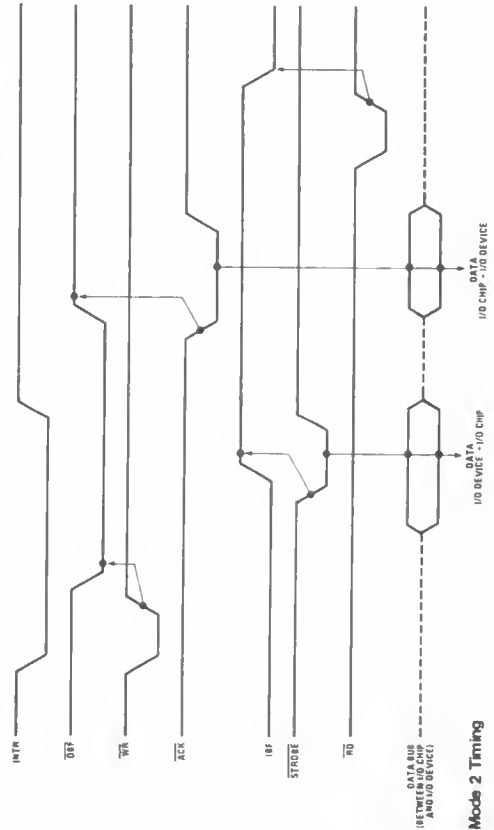
## Operating Modes (cont'd.)

### Mode 2 (Strobed Bidirectional Bus Input/Output)

This mode enables communication with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus input/output). "Handshaking" signals are provided to maintain proper bus flow discipline in a manner similar to Mode 1. In addition, interrupt generation and enable/disable functions are available in Mode 2.

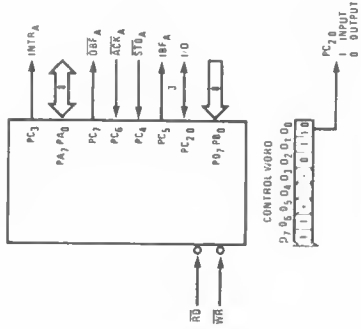


Mode 2

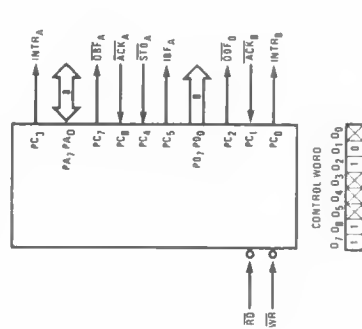


## Operating Modes (cont'd.)

MODE 2 AND MODE 0 (INPUT)



MODE 2 AND MODE 1 (OUTPUT)



Mode 2 Combinations

# Z80<sup>®</sup>-CPU Z80A-CPU



## Product Specification MARCH 1978

The Zilog Z80 product line is a complete set of microcomputer components, development systems and support software. The Z80 microcomputer component set includes all of the circuits necessary to build high-performance microcomputer systems with virtually no other logic and a minimum number of low cost standard memory elements.

The Z80 and Z80A CPU's are third generation single chip microprocessors with unrivaled computational power. This increased computational power results in higher system through-put and more efficient memory utilization when compared to second generation microprocessors. In addition, the Z80 and Z80A CPU's are very easy to implement into a system because of their single voltage requirement plus all output signals are fully decoded and timed to control standard memory or peripheral circuits. The circuit is implemented using an N-channel, ion implanted, silicon gate MOS process.

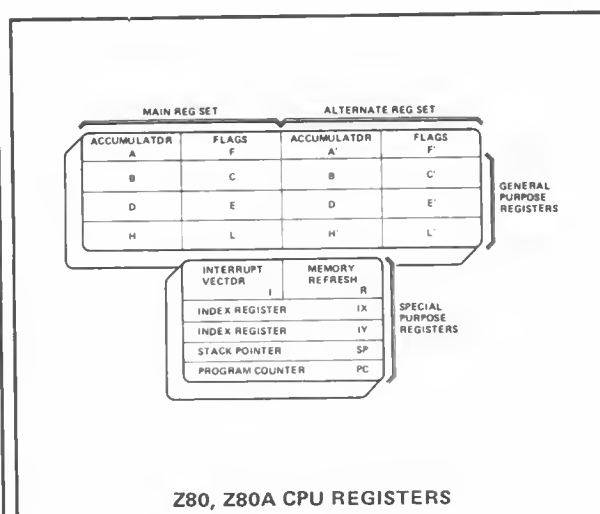
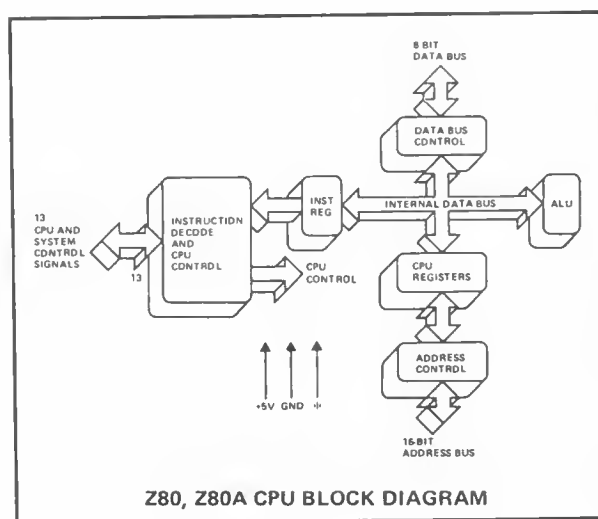
Figure 1 is a block diagram of the CPU, Figure 2 details the internal register configuration which contains 208 bits of Read/Write memory that are accessible to the programmer. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or as 16-bit register pairs. There are also two sets of accumulator and flag registers. The programmer has access to either set of main or alternate registers through a group of exchange instructions. This alternate set allows foreground/background mode of operation or may be reserved for very fast Interrupt response. Each CPU also contains a 16-bit stack pointer which permits simple implementation of

multiple level interrupts, unlimited subroutine nesting and simplification of many types of data handling.

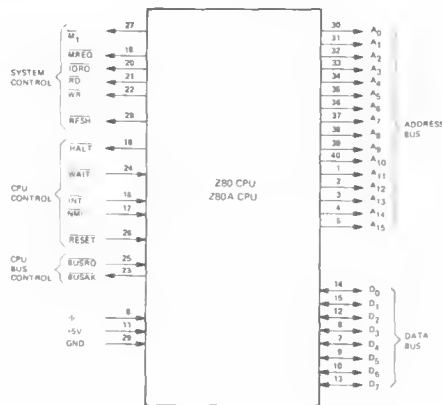
The two 16-bit index registers allow tabular data manipulation and easy implementation of relocatable code. The Refresh register provides for automatic, totally transparent refresh of external dynamic memories. The I register is used in a powerful interrupt response mode to form the upper 8 bits of a pointer to a interrupt service address table, while the interrupting device supplies the lower 8 bits of the pointer. An indirect call is then made to this service address.

### FEATURES

- Single chip, N-channel Silicon Gate CPU.
- 158 instructions—includes all 78 of the 8080A instructions with total software compatibility. New instructions include 4-, 8- and 16-bit operations with more useful addressing modes such as indexed, bit and relative.
- 17 internal registers.
- Three modes of fast interrupt response plus a non-maskable interrupt.
- Directly interfaces standard speed static or dynamic memories with virtually no external logic.
- 1.0  $\mu$ s instruction execution speed.
- Single 5 VDC supply and single-phase 5 volt Clock.
- Out-performs any other single chip microcomputer in 4-, 8-, or 16-bit applications.
- All pins TTL Compatible
- Built-in dynamic RAM refresh circuitry.



## Z80, Z80A-CPU Pin Description



**Z80, Z80A CPU PIN CONFIGURATION**

**A<sub>0</sub>-A<sub>15</sub>**  
(Address Bus) Tri-state output, active high. A<sub>0</sub>-A<sub>15</sub> constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges.

**D<sub>0</sub>-D<sub>7</sub>**  
(Data Bus) Tri-state input/output, active high. D<sub>0</sub>-D<sub>7</sub> constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

**M<sub>1</sub>**  
(Machine Cycle one) Output, active low.  $\overline{M_1}$  indicates that the current machine cycle is the OP code fetch cycle of an instruction execution.

**MREQ**  
(Memory Request) Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

**IORQ**  
(Input/Output Request) Tri-state output, active low. The  $\overline{IORQ}$  signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An  $\overline{IORQ}$  signal is also generated when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus.

**RD**  
(Memory Read) Tri-state output, active low.  $\overline{RD}$  indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

**WR**  
(Memory Write) Tri-state output, active low.  $\overline{WR}$  indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

**RFSH**  
(Refresh) Output, active low.  $\overline{RFSH}$  indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current MREQ signal should be used to do a refresh read to all dynamic memories.

**HALT**  
(Halt state) Output, active low.  $\overline{HALT}$  indicates that the CPU has executed a HALT software instruction and is awaiting either a non-maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.

**WAIT**  
(Wait) Input, active low.  $\overline{WAIT}$  indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active.

**INT**  
(Interrupt Request) Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled.

**NMI**  
(Non Maskable Interrupt) Input, active low. The non-maskable interrupt request line has a higher priority than  $\overline{INT}$  and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. NMI automatically forces the Z-80 CPU to restart to location 0066H.

**RESET** Input, active low.  $\overline{RESET}$  initializes the CPU as follows: reset interrupt enable flip-flop, clear PC and registers I and R and set interrupt to 8080A mode. During reset time, the address and data bus go to a high impedance state and all control output signals go to the inactive state.

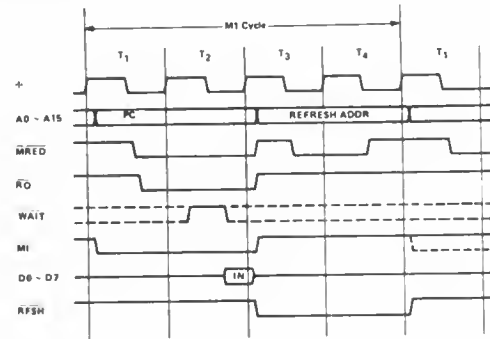
**BUSRQ**  
(Bus Request) Input, active low. The bus request signal has a higher priority than  $\overline{NMI}$  and is always recognized at the end of the current machine cycle and is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these busses.

**BUSAK**  
(Bus Acknowledge) Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.

## Timing Waveforms

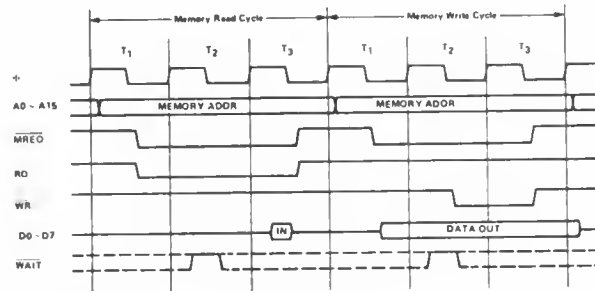
### INSTRUCTION OP CODE FETCH

The program counter content (PC) is placed on the address bus immediately at the start of the cycle. One half clock time later  $\overline{\text{MREQ}}$  goes active. The falling edge of  $\overline{\text{MREQ}}$  can be used directly as a chip enable to dynamic memories.  $\overline{\text{RD}}$  when active indicates that the memory data should be enabled onto the CPU data bus. The CPU samples data with the rising edge of the clock state  $T_3$ . Clock states  $T_3$  and  $T_4$  of a fetch cycle are used to refresh dynamic memories while the CPU is internally decoding and executing the instruction. The refresh control signal  $\overline{\text{RFSH}}$  indicates that a refresh read of all dynamic memories should be accomplished.



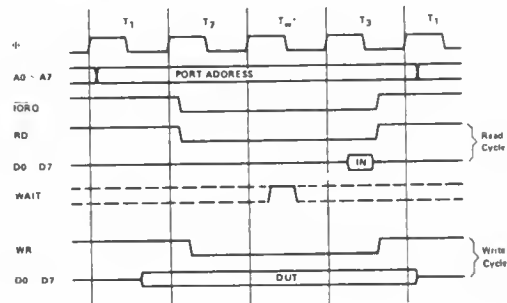
### MEMORY READ OR WRITE CYCLES

Illustrated here is the timing of memory read or write cycles other than an OP code fetch ( $M_1$  cycle). The  $\overline{\text{MREQ}}$  and  $\overline{\text{RD}}$  signals are used exactly as in the fetch cycle. In the case of a memory write cycle, the  $\overline{\text{MREQ}}$  also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The  $\overline{\text{WR}}$  line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory.



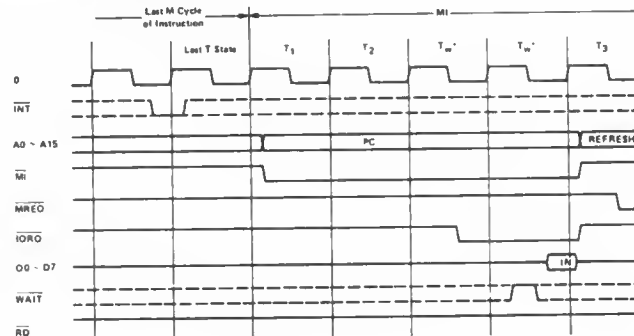
### INPUT OR OUTPUT CYCLES

Illustrated here is the timing for an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted ( $T_{w^*}$ ). The reason for this is that during I/O operations this extra state allows sufficient time for an I/O port to decode its address and activate the  $\overline{\text{WAIT}}$  line if a wait is required.



### INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

The interrupt signal is sampled by the CPU with the rising edge of the last clock at the end of any instruction. When an interrupt is accepted, a special  $M_1$  cycle is generated. During this  $M_1$  cycle, the  $\overline{\text{IORQ}}$  signal becomes active (instead of  $\overline{\text{MREQ}}$ ) to indicate that the interrupting device can place an 8-bit vector on the data bus. Two wait states ( $T_{w^*}$ ) are automatically added to this cycle so that a ripple priority interrupt scheme, such as the one used in the Z80 peripheral controllers, can be easily implemented.



## A.C. Characteristics

## Z80-CPU

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = +5V \pm 5\%$ , Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
$\phi$	$t_c$	Clock Period	4	1121	$\mu\text{sec}$	
	$t_{w(\phi H)}$	Clock Pulse Width, Clock High	180	17	nsec	
	$t_{w(\phi L)}$	Clock Pulse Width, Clock Low	180	2000	nsec	
	$t_{r, f}$	Clock Rise and Fall Time		30	nsec	
$A_{0-15}$	$t_{D(AD)}$	Address Output Delay		145	nsec	$C_L = 50\text{pF}$
	$t_{F(AD)}$	Delay to Float		110	nsec	
	$t_{acnt}$	Address Stable Prior to $\overline{\text{MREQ}}$ (Memory Cycle)	11		nsec	
	$t_{act}$	Address Stable Prior to $\overline{\text{IORQ}}$ , $\overline{\text{RD}}$ or $\overline{\text{WR}}$ (I/O Cycle)	12		nsec	
	$t_{ca}$	Address Stable from $\overline{\text{RD}}$ , $\overline{\text{WR}}$ , $\overline{\text{IORQ}}$ or $\overline{\text{MREQ}}$	13		nsec	
	$t_{cal}$	Address Stable From $\overline{\text{RD}}$ or $\overline{\text{WR}}$ During Float	14		nsec	
$D_{0-7}$	$t_{D(D)}$	Data Output Delay		230	nsec	$C_L = 50\text{pF}$
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{S\phi(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	60		nsec	
	$t_{dcm}$	Data Stable Prior to $\overline{\text{WR}}$ (Memory Cycle)	15		nsec	
	$t_{dci}$	Data Stable Prior to $\overline{\text{WR}}$ (I/O Cycle)	16		nsec	
	$t_{cdi}$	Data Stable From $\overline{\text{WR}}$	17		nsec	
	$t_H$	Any Hold Time for Setup Time	0		nsec	
$\overline{\text{MREQ}}$	$t_{DL\phi}(\text{MR})$	$\overline{\text{MREQ}}$ Delay From Falling Edge of Clock, $\overline{\text{MREQ}}$ Low		100	nsec	$C_L = 50\text{pF}$
	$t_{DH\phi}(\text{MR})$	$\overline{\text{MREQ}}$ Delay From Rising Edge of Clock, $\overline{\text{MREQ}}$ High		100	nsec	
	$t_{DL\phi}(\text{MR})$	$\overline{\text{MREQ}}$ Delay From Falling Edge of Clock, $\overline{\text{MREQ}}$ High		100	nsec	
	$t_{w(\text{MRL})}$	Pulse Width, $\overline{\text{MREQ}}$ Low	18		nsec	
	$t_{w(\text{MRH})}$	Pulse Width, $\overline{\text{MREQ}}$ High	19		nsec	
$\overline{\text{IORQ}}$	$t_{DL\phi}(\text{IR})$	$\overline{\text{IORQ}}$ Delay From Rising Edge of Clock, $\overline{\text{IORQ}}$ Low		90	nsec	$C_L = 50\text{pF}$
	$t_{DL\phi}(\text{IR})$	$\overline{\text{IORQ}}$ Delay From Falling Edge of Clock, $\overline{\text{IORQ}}$ Low		110	nsec	
	$t_{DH\phi}(\text{IR})$	$\overline{\text{IORQ}}$ Delay From Rising Edge of Clock, $\overline{\text{IORQ}}$ High		100	nsec	
	$t_{DH\phi}(\text{IR})$	$\overline{\text{IORQ}}$ Delay From Falling Edge of Clock, $\overline{\text{IORQ}}$ High		110	nsec	
$\overline{\text{RD}}$	$t_{DL\phi}(\text{RD})$	$\overline{\text{RD}}$ Delay From Rising Edge of Clock, $\overline{\text{RD}}$ Low		100	nsec	$C_L = 50\text{pF}$
	$t_{DL\phi}(\text{RD})$	$\overline{\text{RD}}$ Delay From Falling Edge of Clock, $\overline{\text{RD}}$ Low		130	nsec	
	$t_{DH\phi}(\text{RD})$	$\overline{\text{RD}}$ Delay From Rising Edge of Clock, $\overline{\text{RD}}$ High		100	nsec	
	$t_{DH\phi}(\text{RD})$	$\overline{\text{RD}}$ Delay From Falling Edge of Clock, $\overline{\text{RD}}$ High		110	nsec	
$\overline{\text{WR}}$	$t_{DL\phi}(\text{WR})$	$\overline{\text{WR}}$ Delay From Rising Edge of Clock, $\overline{\text{WR}}$ Low		80	nsec	$C_L = 50\text{pF}$
	$t_{DL\phi}(\text{WR})$	$\overline{\text{WR}}$ Delay From Falling Edge of Clock, $\overline{\text{WR}}$ Low		90	nsec	
	$t_{DH\phi}(\text{WR})$	$\overline{\text{WR}}$ Delay From Falling Edge of Clock, $\overline{\text{WR}}$ High		100	nsec	
	$t_{w(\text{WRL})}$	Pulse Width, $\overline{\text{WR}}$ Low	110		nsec	
$\overline{\text{M1}}$	$t_{DL}(\text{M1})$	$\overline{\text{M1}}$ Delay From Rising Edge of Clock, $\overline{\text{M1}}$ Low		130	nsec	$C_L = 50\text{pF}$
	$t_{DH}(\text{M1})$	$\overline{\text{M1}}$ Delay From Rising Edge of Clock, $\overline{\text{M1}}$ High		130	nsec	
$\overline{\text{RFSH}}$	$t_{DL}(\text{RF})$	$\overline{\text{RFSH}}$ Delay From Rising Edge of Clock, $\overline{\text{RFSH}}$ Low		180	nsec	$C_L = 50\text{pF}$
	$t_{DH}(\text{RF})$	$\overline{\text{RFSH}}$ Delay From Rising Edge of Clock, $\overline{\text{RFSH}}$ High		150	nsec	
$\overline{\text{WAIT}}$	$t_s(\text{WT})$	$\overline{\text{WAIT}}$ Setup Time to Falling Edge of Clock	70		nsec	
$\overline{\text{HALT}}$	$t_D(\text{HT})$	$\overline{\text{HALT}}$ Delay Time From Falling Edge of Clock		300	nsec	$C_L = 50\text{pF}$
$\overline{\text{INT}}$	$t_s(\text{IT})$	$\overline{\text{INT}}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{\text{NMI}}$	$t_w(\text{NML})$	Pulse Width, $\overline{\text{NMI}}$ Low	80		nsec	
$\overline{\text{BUSRQ}}$	$t_s(\text{BQ})$	$\overline{\text{BUSRQ}}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{\text{BUSAK}}$	$t_{DL}(\text{BA})$	$\overline{\text{BUSAK}}$ Delay From Rising Edge of Clock, $\overline{\text{BUSAK}}$ Low		120	nsec	$C_L = 50\text{pF}$
	$t_{DH}(\text{BA})$	$\overline{\text{BUSAK}}$ Delay From Falling Edge of Clock, $\overline{\text{BUSAK}}$ High		110	nsec	
$\overline{\text{RFSH}}$	$t_s(\text{RS})$	$\overline{\text{RFSH}}$ Setup Time to Rising Edge of Clock	90		nsec	
	$t_{F(IC)}$	Delay to Float ( $\overline{\text{MREQ}}$ , $\overline{\text{IORQ}}$ , $\overline{\text{RD}}$ and $\overline{\text{WR}}$ )		100	nsec	
	$t_{mr}$	$\overline{\text{M1}}$ Stable Prior to $\overline{\text{IORQ}}$ (Interrupt Ack.)	111		nsec	

$$[12] \quad t_c = t_{w(\phi H)} + t_{w(\phi L)} + t_f + t_r$$

$$[11] \quad t_{acm} = t_{w(\phi H)} + t_f - 75$$

$$[2] \quad t_{act} = t_c - 80$$

$$[3] \quad t_{ca} = t_{w(\phi L)} + t_f - 40$$

$$[4] \quad t_{cal} = t_{w(\phi L)} + t_f - 60$$

$$[5] \quad t_{dcm} = t_c - 210$$

$$[6] \quad t_{dci} = t_{w(\phi L)} + t_f - 210$$

$$[7] \quad t_{cdf} = t_{w(\phi L)} + t_f - 80$$

$$[8] \quad t_{w(\text{MRL})} = t_c - 40$$

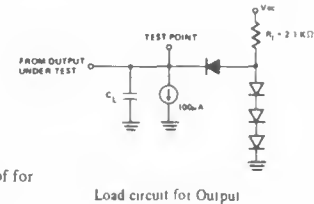
$$[9] \quad t_{w(\text{MRH})} = t_{w(\phi H)} + t_f - 30$$

$$[10] \quad t_{w(\text{WRL})} = t_c - 40$$

$$[11] \quad t_{mr} = 2t_c + t_{w(\phi H)} + t_f - 80$$

### NOTES

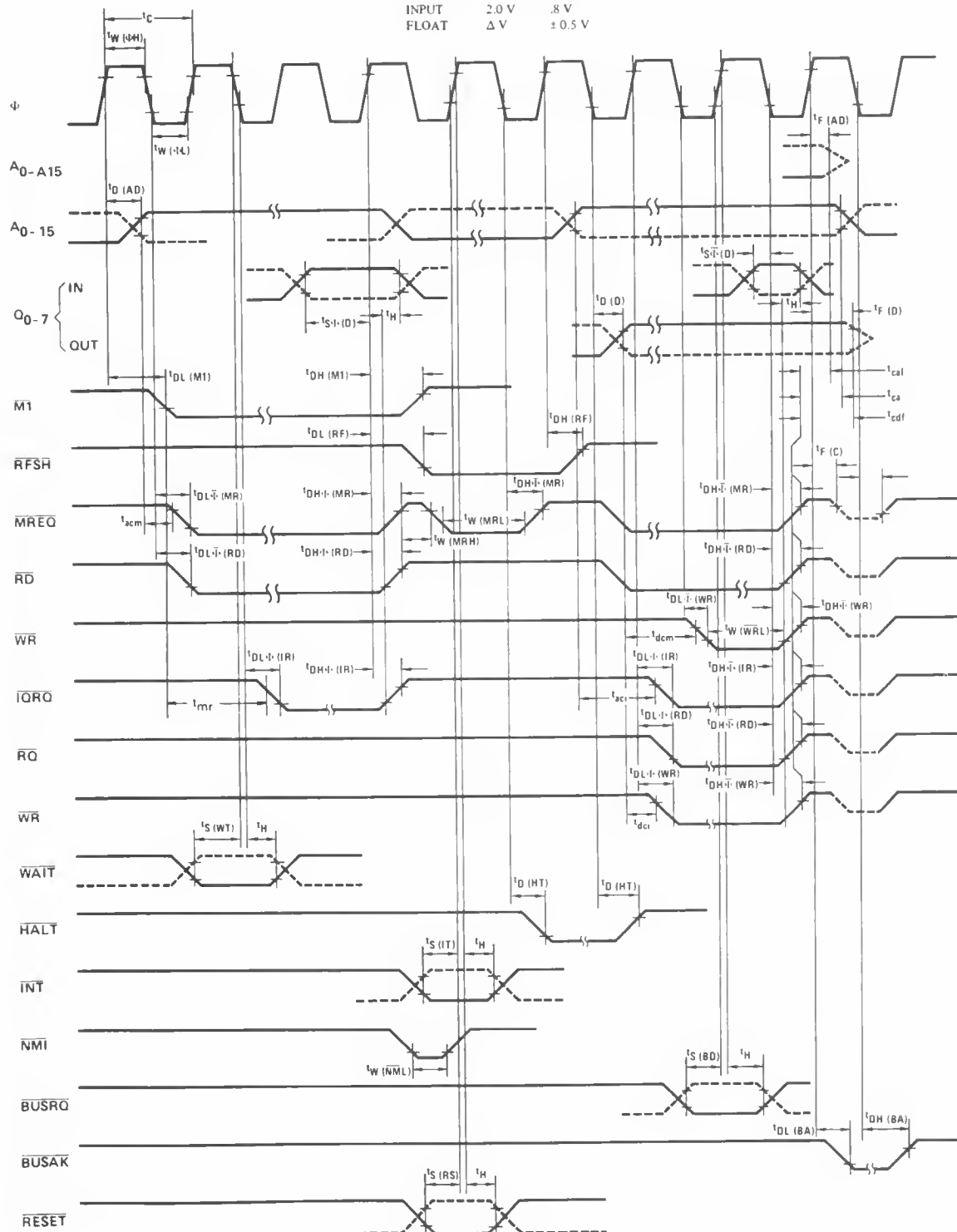
- Data should be enabled onto the CPU data bus when  $\overline{\text{RD}}$  is active. During interrupt acknowledge data should be enabled when  $\overline{\text{M1}}$  and  $\overline{\text{IORQ}}$  are both active.
- All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.
- The  $\overline{\text{RESET}}$  signal must be active for a minimum of 3 clock cycles.
- Output Delay vs. Loaded Capacitance  
 $T_A = 70^\circ\text{C}$      $V_{CC} = +5V \pm 5\%$   
 Add 10nsec delay for each 50pf increase in load up to a maximum of 200pf for the data bus & 100pf for address & control lines.
- Although static by design, testing guarantees  $t_{w(\phi H)}$  of 200 nsec maximum.



## A.C. Timing Diagram

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	$V_{CC} - .6V$	.45V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	$\Delta V$	$\pm 0.5 V$



## Absolute Maximum Ratings

Temperature Under Bias	Specified operating range.
Storage Temperature	-65°C to +150°C
Voltage On Any Pin with Respect to Ground	-0.3V to +7V
Power Dissipation	1.5W

### \*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note For Z80-CPU all AC and DC characteristics remain the same for the military grade parts except  $I_{CC}$ .

$$I_{CC} = 200 \text{ mA}$$

## Z80-CPU D.C. Characteristics

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$  unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
$V_{ILC}$	Clock Input Low Voltage	-0.3		0.45	V	
$V_{IHC}$	Clock Input High Voltage	$V_{CC} - 0.6$		$V_{CC} + 0.3$	V	
$V_{IL}$	Input Low Voltage	-0.3		0.8	V	
$V_{IH}$	Input High Voltage	2.0		$V_{CC}$	V	
$V_{OL}$	Output Low Voltage			0.4	V	$I_{OL} = 1.8 \text{ mA}$
$V_{OH}$	Output High Voltage	2.4			V	$I_{OH} = -250 \mu\text{A}$
$I_{CC}$	Power Supply Current			150	mA	
$I_{LI}$	Input Leakage Current			10	$\mu\text{A}$	$V_{IN} = 0 \text{ to } V_{CC}$
$I_{LOH}$	Tri-State Output Leakage Current in Float			10	$\mu\text{A}$	$V_{OUT} = 2.4 \text{ to } V_{CC}$
$I_{LOL}$	Tri-State Output Leakage Current in Float			-10	$\mu\text{A}$	$V_{OUT} = 0.4 \text{ V}$
$I_{LD}$	Data Bus Leakage Current in Input Mode			$\pm 10$	$\mu\text{A}$	$0 \leq V_{IN} \leq V_{CC}$

## Capacitance

$T_A = 25^\circ\text{C}$ ,  $f = 1 \text{ MHz}$ ,  
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
$C_\Phi$	Clock Capacitance	35	pF
$C_{IN}$	Input Capacitance	5	pF
$C_{OUT}$	Output Capacitance	10	pF

## Z80-CPU

### Ordering Information

C - Ceramic  
P - Plastic

S - Standard  $5V \pm 5\%$   $0^\circ$  to  $70^\circ\text{C}$

E - Extended  $5V \pm 5\%$   $-40^\circ$  to  $85^\circ\text{C}$

M - Military  $5V \pm 10\%$   $-55^\circ$  to  $125^\circ\text{C}$

/

## Z80A-CPU D.C. Characteristics

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$  unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
$V_{ILC}$	Clock Input Low Voltage	-0.3		0.45	V	
$V_{IHC}$	Clock Input High Voltage	$V_{CC} - 0.6$		$V_{CC} + 0.3$	V	
$V_{IL}$	Input Low Voltage	-0.3		0.8	V	
$V_{IH}$	Input High Voltage	2.0		$V_{CC}$	V	
$V_{OL}$	Output Low Voltage			0.4	V	$I_{OL} = 1.8 \text{ mA}$
$V_{OH}$	Output High Voltage	2.4			V	$I_{OH} = -250 \mu\text{A}$
$I_{CC}$	Power Supply Current		90	200	mA	
$I_{LI}$	Input Leakage Current			10	$\mu\text{A}$	$V_{IN} = 0 \text{ to } V_{CC}$
$I_{LOH}$	Tri-State Output Leakage Current in Float			10	$\mu\text{A}$	$V_{OUT} = 2.4 \text{ to } V_{CC}$
$I_{LOL}$	Tri-State Output Leakage Current in Float			-10	$\mu\text{A}$	$V_{OUT} = 0.4 \text{ V}$
$I_{LD}$	Data Bus Leakage Current in Input Mode			$\pm 10$	$\mu\text{A}$	$0 \leq V_{IN} \leq V_{CC}$

## Capacitance

$T_A = 25^\circ\text{C}$ ,  $f = 1 \text{ MHz}$ ,  
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
$C_\Phi$	Clock Capacitance	35	pF
$C_{IN}$	Input Capacitance	5	pF
$C_{OUT}$	Output Capacitance	10	pF

## Z80A-CPU

### Ordering Information

C - Ceramic  
P - Plastic

S - Standard  $5V \pm 5\%$   $0^\circ$  to  $70^\circ\text{C}$

# A.C. Characteristics

# Z80A-CPU

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = +5V \pm 5\%$ , Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
$\phi$	$t_c$	Clock Period	.25	11.21	$\mu\text{sec}$	
	$t_w(\phi H)$	Clock Pulse Width, Clock High	110	[E]	nsec	
	$t_w(\phi L)$	Clock Pulse Width, Clock Low	110	2000	nsec	
	$t_{r,f}$	Clock Rise and Fall Time		30	nsec	
$A_{0-15}$	$t_D(AD)$	Address Output Delay		110	nsec	$C_L = 50\text{pF}$
	$t_F(AD)$	Delay to Float		90	nsec	
	$t_{acm}$	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	111		nsec	
	$t_{aci}$	Address Stable Prior to $\overline{IORQ}$ , $\overline{RD}$ or $\overline{WR}$ (I/D Cycle)	121		nsec	
	$t_{ca}$	Address Stable from $\overline{RD}$ , $\overline{WR}$ , $\overline{IDRQ}$ or $\overline{MREQ}$	131		nsec	
	$t_{caf}$	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	141		nsec	
$D_{0-7}$	$t_D(D)$	Data Output Delay		150	nsec	$C_L = 50\text{pF}$
	$t_F(D)$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\phi}(D)$	Data Setup Time to Rising Edge of Clock During M1 Cycle	35		nsec	
	$t_{S\phi}(D)$	Data Setup Time to Falling Edge of Clock During M2 to M5	50		nsec	
	$t_{dcm}$	Data Stable Prior to $\overline{WR}$ (Memory Cycle)	151		nsec	
	$t_{dci}$	Data Stable Prior to $\overline{WR}$ (I/D Cycle)	161		nsec	
	$t_{cdf}$	Data Stable From $\overline{WR}$	171		nsec	
	$t_H$	Any Hold Time for Setup Time		0	nsec	
$\overline{MREQ}$	$t_{DL\phi}(\overline{MR})$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low		85	nsec	$C_L = 50\text{pF}$
	$t_{DH\phi}(\overline{MR})$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		85	nsec	
	$t_{DH\phi}(\overline{MR})$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		85	nsec	
	$t_w(\overline{MRL})$	Pulse Width, $\overline{MREQ}$ Low	181		nsec	
	$t_w(\overline{MRH})$	Pulse Width, $\overline{MREQ}$ High	191		nsec	
$\overline{IDRQ}$	$t_{DL\phi}(\overline{IR})$	$\overline{IDRQ}$ Delay From Rising Edge of Clock, $\overline{IDRQ}$ Low		75	nsec	$C_L = 50\text{pF}$
	$t_{DL\phi}(\overline{IR})$	$\overline{IDRQ}$ Delay From Falling Edge of Clock, $\overline{IDRQ}$ Low		85	nsec	
	$t_{DH\phi}(\overline{IR})$	$\overline{IDRQ}$ Delay From Rising Edge of Clock, $\overline{IDRQ}$ High		85	nsec	
	$t_{DH\phi}(\overline{IR})$	$\overline{IDRQ}$ Delay From Falling Edge of Clock, $\overline{IDRQ}$ High		85	nsec	
				85	nsec	
$\overline{RD}$	$t_{DL\phi}(\overline{RD})$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		85	nsec	$C_L = 50\text{pF}$
	$t_{DL\phi}(\overline{RD})$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		95	nsec	
	$t_{DH\phi}(\overline{RD})$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		85	nsec	
	$t_{DH\phi}(\overline{RD})$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		85	nsec	
				85	nsec	
$\overline{WR}$	$t_{DL\phi}(\overline{WR})$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		65	nsec	$C_L = 50\text{pF}$
	$t_{DL\phi}(\overline{WR})$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low		80	nsec	
	$t_{DH\phi}(\overline{WR})$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		80	nsec	
	$t_w(\overline{WRL})$	Pulse Width, $\overline{WR}$ Low	1101		nsec	
					nsec	
$\overline{MI}$	$t_{DL}(\overline{MI})$	$\overline{MI}$ Delay From Rising Edge of Clock, $\overline{MI}$ Low		100	nsec	$C_L = 50\text{pF}$
	$t_{DH}(\overline{MI})$	$\overline{MI}$ Delay From Rising Edge of Clock, $\overline{MI}$ High		100	nsec	
$\overline{RFSH}$	$t_{DL}(\overline{RF})$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		130	nsec	$C_L = 50\text{pF}$
	$t_{DH}(\overline{RF})$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		120	nsec	
$\overline{WAIT}$	$t_s(\overline{WT})$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		nsec	
$\overline{HALT}$	$t_D(\overline{HT})$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300	nsec	$C_L = 50\text{pF}$
$\overline{INT}$	$t_s(\overline{IT})$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{NMI}$	$t_w(\overline{NML})$	Pulse Width, $\overline{NMI}$ Low	80		nsec	
$\overline{BUSRQ}$	$t_s(\overline{BQ})$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	50		nsec	
$\overline{BUSA\overline{K}}$	$t_{DL}(\overline{BA})$	$\overline{BUSA\overline{K}}$ Delay From Rising Edge of Clock, $\overline{BUSA\overline{K}}$ Low		100	nsec	$C_L = 50\text{pF}$
	$t_{DH}(\overline{BA})$	$\overline{BUSA\overline{K}}$ Delay From Falling Edge of Clock, $\overline{BUSA\overline{K}}$ High		100	nsec	
$\overline{RESET}$	$t_s(\overline{RS})$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	60		nsec	
	$t_F(C)$	Delay to Float ( $\overline{MREQ}$ , $\overline{IDRQ}$ , $\overline{RD}$ and $\overline{WR}$ )		80	nsec	
	$t_{mr}$	$\overline{MI}$ Stable Prior to $\overline{IORQ}$ (Interrupt Ack.)	1111		nsec	

$$[12] t_c = t_w(\phi H) + t_w(\phi L) + t_r + t_f$$

$$[1] t_{acm} = t_w(\phi H) + t_f - 65$$

$$[2] t_{aci} = t_c - 70$$

$$[3] t_{ca} = t_w(\phi L) + t_f - 50$$

$$[4] t_{caf} = t_w(\phi L) + t_f - 45$$

$$[5] t_{dcm} = t_c - 170$$

$$[6] t_{dci} = t_w(\phi L) + t_f - 170$$

$$[7] t_{cdf} = t_w(\phi L) + t_f - 70$$

$$[8] t_w(\overline{MRL}) = t_c - 30$$

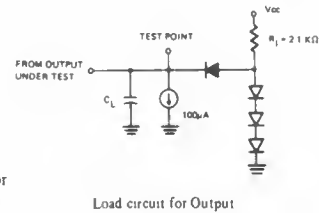
$$[9] t_w(\overline{MRH}) = t_w(\phi H) + t_f - 20$$

$$[10] t_w(\overline{WRL}) = t_c - 30$$

$$[11] t_{mr} = 2t_c + t_w(\phi H) + t_f - 65$$

## NOTES

- Data should be enabled onto the CPU data bus when  $\overline{RD}$  is active. During interrupt acknowledge data should be enabled when  $\overline{MI}$  and  $\overline{IORQ}$  are both active.
- All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.
- The  $\overline{RESET}$  signal must be active for a minimum of 3 clock cycles.
- Output Delay vs. Loaded Capacitance  
 $T_A = 70^\circ\text{C}$   $V_{CC} = +5V \pm 5\%$   
 Add 10nsec delay for each 50pf increase in load up to maximum of 200pf for data bus and 100pf for address & control lines.
- Although static by design, testing guarantees  $t_w(\phi H)$  of 200  $\mu\text{sec}$  maximum



# NOTES

## I GLOSSARY

**ACCESS** The operation of seeking, reading or writing data on a storage unit (in this case, the diskette).

**ACCESS TIME** The time that elapses between any instruction being given to access some data and that data becoming available for use.

**ACTIVE RECORDS TABLE (ART)** A table of binary values in which the relative position of a single value determines the status of a record with the same relative position; i.e., the Nth binary number determines the status of the Nth record. **EXAMPLE:** If the 8th binary number in the table is a zero, then the 8th record is inactive. Conversely, if the 8th binary number in the table is a one, then the 8th record is active.

**ADDRESS** An identification (number, name, or label) for a location in which data is stored.

**ALGORITHM** A computational procedure.

**ALPHANUMERIC (CHARACTERS)** A generic term for numeric digits, alphabetic characters, punctuation characters and special characters.

**ALPHANUMERIC STRING** A group of characters which may include digits, alphabetic characters, punctuation characters and special characters, and may include spaces. (NOTE: a space is a 'character' to the computer, as it must generate a code for spaces as well as symbols.)

**ASCII** Abbreviation for American Standard Code for Information Interchange. Pronounced: Ass-KEY. Usually refers to a standard method of encoding letter, numeral, symbol and special function characters, as used by the computer industry.

**ASSEMBLY LANGUAGE** A machine level language for programming, such as Radio Shack's "EDITOR/ASSEMBLER" which uses Z-80 processor mnemonics and automatically 'assembles' machine readable code from the mnemonics.

**BASE** A quantity of characters for use in each of the digital positions of a numbering system.

**BASE 2** The 'BINARY' numbering system consisting of more than one symbol, representing a sum, in which the individual quantity represented by each figure is based on a multiple of 2.

**BASE 10** The 'DECIMAL' numbering system - consisting of more than one symbol, representing a sum, in which the individual quantity represented by each symbol is based on a multiple of 10.

**BASE 16** The 'HEXADECIMAL' numbering system - consisting of more than one symbol representing a sum, in which the individual quantity represented by each symbol is based on a multiple of 16.

BINARY See 'BASE 2'

BIT A single 'BINARY' digit whose value is 'zero' or 'one'.

BOOLEAN A form of algebra applied to binary numbers which is similar in form to ordinary algebra. It is especially useful for logical analysis of binary numbers as used in computers.

'BOOT' - BOOTSTRAP A machine language program file that is put onto every diskette by the 'FORMAT' routine. This routine is invoked when reset or power-on occurs. It automatically loads the necessary programs (SYS0/SYS) to cause the computer to respond to the DOS commands; i.e., the machine is 'BOOTSTRAPPED' or 'BOOTED' into operation.

BUFFER A small area of memory used for the temporary storage of data to be processed.

BUFFER TRACK A track on a diskette used for the temporary storage of data or program material during a recovery process.

BUG A software fault that results in the malfunction of a program. May also refer to hardware malfunctions.

BYTE Eight 'BITS'. A 'BYTE' may represent any numerical value between '0' and '255'.

CLOBBERED A slang term referring to the non-operation of software, hardware, computer device, or storage media (such as disk) usually as the result of a program or hardware error.

COMMAND FILE A file consisting of a list of commands, to be executed in sequence.

CONTIGUOUS Adjacent or adjoining.

CONTROL CODE In programming, instructions which determine conditional jumps are often referred to as control instructions and the time sequence of execution of instructions is called the flow of control.

CRC ERROR Cyclic Redundancy Check. A means of checking for errors by using redundant information used primarily to check disk I/O on the TRS-80.

DATA BASE A collection of interrelated data stored together with controlled redundancy to serve one or more applications. The data are stored so that they are independent of programs which use the data. A common and controlled approach is used in adding new data and in modifying and retrieving existing data within a data base. A system is said to contain a collection of data-based information if they are disjoint in structure.

DATA BASE MANAGEMENT SYSTEM      The collection of software required for using a data base.

DATA ELEMENT      Synonymous with 'DATA ITEM' or 'FIELD'

DATA TYPE      The form in which data is stored; i.e., integer, single precision, double precision, 'alphanumeric' character strings or 'strings'.

DEC      Initials for Directory Entry Code.

DECIMAL      See 'BASE 10'.

DIRECT ACCESS      Retrieval or storage of data by a reference to its location on a disk, rather than relative to the previously retrieved or stored data.

DIRECT STATEMENT (IN FILE)      A program statement that exists in the disk file that is not assigned a line number.

DIRECTORY      A table giving the relationships between items of data. Sometimes a table or an index giving the addresses of data.

DISPLACEMENT      A specified number of sectors, at the top or beginning of the file, in which the 'bookkeeping' and file parameters are stored for later use by the various program modules.

DISTRIBUTED FREE SPACE      Space left empty at intervals in a data layout to permit the possible insertion of new data.

DOUBLE PRECISION      A positive or negative numeric value, 16 digits in length, not including a decimal point (EXAMPLE: 99999999999999.99).

DUMP      To transfer all or part of the contents of one section of computer memory or disk into another section, or to some other computer device.

DYNAMIC STORAGE ALLOCATION      The allocation of storage space by a procedure based on the instantaneous or actual demand for storage space by that procedure, rather than allocating storage space to a procedure based on its anticipated or predicted demand.

EATEN (DIRECTORY/DISK)      Slang term. See 'CLOBBERED'.

EMBEDDED POINTERS      Pointers in the data records rather than in a directory.

ENTITY      Something about which data is recorded.

EOF      Initials for 'END OF FILE'. It is common practice to say that the EOF is record number nn or that the EOF is byte 15 of sector 12. Hence, it is a convenient term to use in describing the location of the last record or last byte in a file.

EXTENT    A contiguous area of data storage.

FILE    A collection of related records treated as a unit; The word file is used in the general sense to mean any collection of informational items similar to one another in purpose, form and content.

FILE PARAMETERS    The data that describes or defines the structure of the file.

FILESPEC    A file specification and may include the 'FILE NAME', 'FILE NAME EXTENSION', 'PASSWORD', and 'DISK DRIVE' specification.

FIELD    See 'DATA ITEM'.

FLAKY    Slang term -Alludes to less than acceptable performance.

FILE AREA    The physical location of the file, on the disk, or in memory.

'FPDE'    Initials for File Primary Directory Entry; a file's entry and file area pointers in the disk directory.

'FXDE'    Initials for File Extended Directory Entry; a file's entry and file area pointers, in the case of an overflow in the 'FPDE'.

GAT    Initials for Granule Allocation Table; A table from which available file areas are assigned to file entries.

GRANULE    Unit of 5 sectors. On the TRS-80 disk operating system, a 'granule' is the basic unit of disk storage allocation. The diskette 'DIRECTORY' file keeps track of free and assigned disk space in terms of 'granules'.

HASH CODE    A code number generated and used as a direct addressing technique in which the key is converted to a pseudo-random number from which the required address is derived.

HEADER RECORD    A record containing common, constant or identifying information for a group of records which follow.

HEXADECIMAL    See 'BASE 16'

HIT    Initials for Hash Index Table; an addressing technique in which a disk file is referenced by a code number in a table, and the position of that code in the table relates to the file entry in the directory.

INDEX    A table used to determine the location of a record.

INDIRECT ADDRESSING    Any method of specifying or locating a storage location whereby the key (of itself or through calculation) does not represent an address. For example, locating an address through indices.

**INSTRING** (INSTRING SEARCH) Refers to the capability of locating a substring of characters that may exist in another character string. An example would be: Substring = "THE" String = "NOW IS THE TIME". An INSTRING routine would locate the substring and return its starting position within that string. In this example, it would return a value of eight.

**INTEGER** A natural or whole number. In the TRS-80, integer values may not exceed the range of +32767 to -32768.

**INVERTED FILE** A file structure which permits fast spontaneous searching for previous unspecified information. Independent lists or indices are maintained in records' keys which are accessible according to the values of specific fields.

**INVERTED LIST** A list organized by a secondary key --- not a primary key.

**IPL** Initials for Initial Program Loader; a program usually executed upon pressing of the 'RESET' button.

**KEY** A data item used to identify or locate a record or other data grouping.

**LABEL** A set of symbols used to identify or describe an item, record, message or file. Occasionally, it may be the same as the address in storage.

**LEAST SIGNIFICANT BYTE** The significant byte contributing the smallest quantity to the value of a numeral.

**LIST** An ordered set of data items. A 'chain'.

**LOAD MODULE** A program developed for loading into storage and being executed when control is passed to the program.

**LOCK-OUT (TRACKS)** Unusable tracks, on the disk, that are not accessible because of damage or by user option.

**LOGICAL** An adjective describing the form of data organization, hardware or system that is perceived by an application program, programmer, or user; it may be different than the real (PHYSICAL) form.

**LOGICAL DATA-BASE DESCRIPTION** A schema. A description of the overall data-base structure, as perceived for the users, which is employed by the data base management software.

**LOGICAL FILE** A file as perceived by an application program; it may be in a completely different form from that in which it is stored on the storage units.

**LOGICAL OPERATOR** A mathematical symbol that represents a mathematical process to be performed on an associated operand. Such operators are 'AND', 'OR', 'NOT', 'AND NOT' and 'OR NOT'.

**LOGICAL RECORD** A record or data item as perceived by an application program; it may be in a completely different form from that in which it is stored on the storage units.

**LSB** See LEAST SIGNIFICANT BYTE.

**MACHINE LANGUAGE** Direct machine readable code.

**MAINTENANCE OF A FILE** (1) The addition, deletion, changing or updating of records in the database. (2) Periodic reorganization of a file to better accommodate items that have been added.

**MONITOR** A program that may supervise the operation of another program for operation or debugging or other purposes.

**MOST SIGNIFICANT BYTE** The significant byte contributing the greatest quantity to the value of a numeral.

**MSB** See MOST SIGNIFICANT BYTE.

**MULTIPLE-KEY RETRIEVAL** Retrieval which requires searches of data based on the values of several key fields (some or all of which are secondary keys).

**NULL** An absence of information as contrasted with zero or blank for the presence of no information.

**NYBBLE** The four right most or left most binary digits of a byte.

**ON-LINE** An on-line system is one in which the input data enter the computer directly from their point of origin, and/or output data are transmitted directly to where they are used. The intermediate stages such as writing tape, loading disks or off-line printing are avoided.

**ON-LINE STORAGE** Storage devices and especially the storage media which they contain under the direct control of a computing system, not off-line or in a volume library.

**OPEN RECORDS TABLE (ORT)** A table of binary values in which the relative position of a single value determines the status of a record with the same relative position; i.e., the Nth binary number determines the status of the Nth record. EXAMPLE: If the 8th binary number in the table is a zero, then the 8th record is open. Conversely, if the 8th binary number in the table is a one, then the 8th record is on file.

**OPERATING SYSTEM** Software which enables a computer to supervise its own operations, automatically calling in programs, routines, language and data as needed for continuous throughput of different types of jobs.

**PARITY** Parity relates to the maintenance of a sameness of level or count, i.e., keeping the same number of binary ones in a computer word and thus be able to perform a check based on an even or odd number for all words under examination.

**PHYSICAL** An adjective, contrasted with logical, which refers to the form in which data or systems exist in reality. Data is often converted by software from the form in which it is physically stored to a form in which a user or programmer perceives it.

**PHYSICAL DATA BASE** A data base in the form in which it is stored on the storage media, including pointers or other means of interconnecting it. Multiple logical data bases may be derived from one or more physical data bases.

**PHYSICAL RECORD** A collection of bits that are physically recorded on the storage medium and which are read or written by one machine input/output instruction.

**POINTER** The address or a record (or other data groupings) contained in another record so that a program may access the former record when it has retrieved the latter record. The address can be absolute, relative or symbolic, hence, the pointer is referred to as absolute, relative or symbolic.

**PRIMARY ENTRY** The main entry made to the directory. Also see 'FPDE'.

**RANDOM ACCESS** To obtain data directly from any storage location regardless of its position, with respect to the previously referenced information. Also called 'DIRECT ACCESS'.

**RANDOM ACCESS STORAGE** A storage technique in which the time required to obtain information is independent of the location of the information most recently obtained.

**READ** To accept or copy information or data from input devices or a memory register; i.e., to read out, to read in.

**RECORD** A group of related fields of information treated as a unit by an application program.

**RELATIONAL OPERATOR** A mathematical symbol that represents a mathematical process to perform a comparison describing the relationship between two values (< less than....> greater than... = equal.... <> not equal... and combinations thereof (see TRS-80 LEVEL II manual, Section 1, Page 5). On the TRS-80, relational comparisons may be made on string values as well as numerical values.

**RELATIVE (as pertains to position)** An address or position that is referenced to a point of origin; i.e. X+20 is a specific position, 20 places from the reference point. If the reference point was at 50, then the absolute position would be at 70 (50+20=70). Also, 50 (since it is the starting reference point) is at relative position 0.

**SCHEMA** A map of the overall logical structure of a database.

**SEARCH** To examine a series of items for any that have a desired property or properties.

**SECONDARY INDEX** An index composed of secondary keys rather than primary keys.

**SECTOR** The smallest addressable portion of storage on a diskette (a unit of 256 bytes on a TRS-80 diskette).

**SEEK** To position the access mechanism of a direct-access storage device at a specified location.

**SEQUENTIAL ACCESS** Access in which records must be read serially or sequentially one after the other; i.e., ASCII files, tape.

**SINGLE PRECISION** A positive or negative numerical value of 6 digits in length, not including a decimal point (EXAMPLE: 99999.9).

**SORT** To arrange a file or data in a sequence by a specified key (may be alphabetic or numeric and in descending or ascending order).

**SOURCE CODE** The text from which code that may be executed is derived.

**SYSTEM FILE** A program used by the operating system to manage the executing program and/or the computer's resources.

**SUB-STRINGS SUB-STRING SEARCH** See INSTRING

**TABLE** A collection of data suitable for quick reference, each item being uniquely identified either by a label or its relative position.

**TALLY** To add or subtract a digit from a quantity.

**TOKEN** A one byte code representing a larger word consisting of 2 or more characters.

**TRACK** The circular recording surface traversed by a read/write head on the disk. On the TRS-80 a track contains 10 sectors (2 granules).

**TRANSACTION** An input record applied to an established file. The input record describes some "event" that will either cause a new file record to be generated, an existing record to be changed or an existing record to be deleted.

**TRANSPARENT** Complexities that are hidden from the programmers or users (made transparent to them) by the software.

**VECTOR** A line representing the properties of magnitude and direction. Since such a 'line' can be described in mathematical terms, a mathematical description (expressed in numbers, of course) of a given 'direction' and 'magnitude' is referred to as a "vector".

**VERIFY** To check a data transfer or transcription.

**WORKING STORAGE** A portion of storage, usually computer main memory, reserved for the temporary results of operations.

**WRITE** To record information on a storage device.

**ZAP** To change a byte or bytes of data in memory or on diskette by using a software utility program.

**ZEROETH** Zeroeth is to '0' as first is to '1'; in computer terms the first position of anything is usually described as the 'zeroeth' and the next position is the 'first' and so on.

# NOTES

PARTS LIST  
SIMPLE HOBBYIST INTERFACE

IC	TYPE	+5 VOLTS	GROUND
Z1	81LS95	20	10
Z2	81LS96	20	10
Z3	81LS95	20	10
Z4	81LS95	20	10
Z5	74LS75	5	12
Z6	74LS75	5	12
Z7	74LS30	14	7
Z8	74LS02	14	7

PARTS LIST  
ONE-SECOND INTERRUPT REAL-TIME CLOCK

IC	TYPE	+5 VOLTS	GROUND
Z1	7805	3 (OUTPUT)	2
Z2	74LS14	14	7
Z3	74LS90	5	10
Z4	74LS92	5	10
Z5	74LS74	14	7
Z6	74LS75	5	12

PART	VALUE	NOTES
C1	4700 mf, 16 volts	
C2	220 mf, 16 volts	
C3	100 nf (0.1 mf)	
C4	100 nf (0.1 mf)	

PARTS LIST  
MSM5832 REAL-TIME CLOCK/CALENDAR

IC	TYPE	+5 VOLTS	GROUND
Z1	74LS30	14	7
Z2	74LS260	14	7
Z3	INS8255	26	7
Z4	MSM5832	1	13

PART	VALUE	NOTES
C1	20 pf	
C2	20 pf	
C3	100 nf (0.1 mf)	
C4	100 nf (0.1 mf)	
C5	100 nf (0.1 mf)	
C6	100 nf (0.1 mf)	
R1-R12	10k (10,000 ohms)	

Battery backup:

PART	VALUE	NOTES
C6	470 mf, 16 volts	
C7	10 mf, 16 volts	
C8	100 nf (0.1 mf)	
C9	10 mf, 16 volts	
O1	Bridge rectifier, 1A 50V	
O2	1N914 or equivalent	
Q1	PNP transistor, Vce=0.1 volt	
Q2	NPN transistor	
R13	47k (47,000 ohms)	
R14	10k (10,000 ohms)	
R15	10k (10,000 ohms)	
R16	100R (100 ohms), 0.5 W	
T1	6V3 (6.3 volt), 1A transformer	
Z5	7805 5-volt regulator	

PARTS LIST  
QUAD SOUND, 800PS AND SLEEPS GENERATOR

IC	TYPE	+5 VOLTS	GROUND
Z1	74LS374	20	10
Z2	74LS125	14	7
Z3	74LS30	14	7
Z4	74LS30	14	7
Z5	7LS02	14	7

PART	VALUE	NOTES
R1-R5	1k (10,000 ohms)	

PARTS LIST  
MEMORY SIDECAR ROM AND RAM ADDITION

IC	TYPE	+5 VOLTS	GROUND
Z1	74LS30	14	7
Z2	74LS02	14	7
Z3	74LS00	14	7
Z4	74LS00	14	7
Z5	74LS125	14	7
Z6,7,8,9	2114-AN4L	18	9
Z10	2716	24	12

PART	VALUE	NOTES
R1-R16	1kD (10,000 ohms)	

PARTS LIST  
MUSIC SYNTHESIZER INTERFACE BOARD

IC	TYPE	+5 VOLTS	GROUND
Z1	81LS95	20	10
Z2	74LS154	24	12
Z3	74LS00	14	7
Z4	74LS04	14	7
Z5	INS8255	26	7
Z6	MC1408L8 OR DAC0808	13	2
Z7	MC1408L8 OR DAC0808	13	2
Z8	74LS123	14	7
Z9	74LS123	14	7
Z10	LM324	4	11

Note that Z6 and Z7 also require -15 volts on pin number 3 and end +7.5 volt reference at pin 14 (through a variable resistor).

PART	VALUE	NOTES
C1,2	20 pf	
C3,4,5,6	100 nf (0.1 mf) (may be changed)	
R1,3	4k7 (4700 ohms)	
R2,4	1kD (1000 ohms) (may be changed)	
R5,6,7,8	1MD (1,000,000 ohms)	

PARTS LIST  
BANK-SELECT ROM/RAM ADDITION

IC	TYPE	+5 VOLTS	GROUND
Z1	81LS95	20	10
Z2	81LS95	20	10
Z3	81LS95	20	10
Z4	81LS95	20	10
Z5-11	2716 OR 4118	24	12
Z12	74LS20	14	7
Z13	74LS260	14	7
Z14	74LS02	14	7
Z15	74LS75		
Z16	74154	24	12
Z17	74LS04	14	7
Z18-26	2716 OR 4118	24	12

PARTS LIST  
6-TRACK MASS STORAGE SYSTEM

IC	TYPE	+5 VOLTS	+12 VOLTS	GROUND
Z1	70C98/60C98	.	.	.
Z2	74LS04	14	.	7
Z3	74LS125	14	.	7
Z4	LF353	.	.	.
Z5	LM339	.	.	.
Z6	70C98/80C98	.	.	.
Z7	LF353	.	.	.
Z8	74LS00	14	.	7
Z9	75452	8	.	4
Z10	70C96/80C96	.	.	.
Z11	75452	8	.	4
Z12	74LS373	20	.	10
Z13	74LS30	14	.	7
Z14	74LS04	14	.	7

PART	VALUE	NOTES
C1	100 pf	
C2	220 pf	
C3	220 nf (0.22 mf)	
C4	10 mf, 16 volts	
C5	100 pf	
C6	220 pf	
C7	220 nf (0.22 mf)	
C8	10 mf, 16 volts	
C9,10,11	100 nf (0.1 mf)	
C12	470 mf, 35 volts	
O1,2,3,4,5	1N914 or equivalent	
K1,2,3	Miniature SPST 5-volt relay	
R1,2,14,17, 18,30,38,39, 40,41	10k (10,000 ohms)	
R3,4,12,19, 20,28	470R (470 ohms)	
R5,6,9,11,16, 22,25,27,32	2k2 (2200 ohms)	
R7,8,23,24	220k (220,000 ohms)	
R10,26	100k (100,000 ohms)	
R13,29	22k (22,000 ohms)	
R15,31	220R (220 ohms)	
R33,34,35,36, 37,42	1k0 (1000 ohms)	
R43	100R (100 ohms)	
R44	75R (75 ohms)	May need adjustment
S1	Carriage-in-place leaf switch	
S2	Track change light switch	
S3	Split-pole foil sensor switch	

PARTS LIST  
MICRO FRONT PANEL MONITOR

IC	TYPE	+5 VOLTS	GROUND
Z1	74LS373	20	10
Z2	74LS373	20	10
Z3	74LS373	20	10
Z4	74LS20	14	7

PART	VALUE	NOTES
DIS1,2,3	10-segment bar LED	
LED1-24	Subminiature LED	
R1-24	270R (270 ohms)	
R25,26,27,28	1k0 (1000 ohms)	
S1	4-position DIP switch	

PARTS LIST  
4K DYNAMIC RAM ADDITION

IC	TYPE	+5 VOLTS	GROUND
Z1	81LS95	20	10
Z2	74LS157	16	8
Z3	81LS95	20	10
Z4	74LS157	16	8
Z5	81LS95	20	10
Z6-13	MK4114 4K RAMS	9	16
Z14	74LS20	14	7
Z15	74LS86	14	7
Z16	81LS95	20	10

Note that Z6 through Z13 also require -5 volts on pin number 1 and +12 volts on pin number 8.

PART	VALUE	NOTES
R1,2,3	1k0 (1000 ohms)	
S1	4-position DIP switch	

PARTS LIST  
HIGH-SPEED, REVERSE VIDEO, UPPER/LOWER CASE,  
INDIVIDUAL REVERSE VIDEO MODES

IC	TYPE	+5 VOLTS	GROUND
P45 (ZMEM)	2102-4L	10	9
P25 (ZBITS)	74LS10	14	7
P6 (ZFAZE)	74LS04	14	7
P27 (ZMODE)	74LS368	20	10
P24 (ZMUXX)	74LS86	14	7
P26 (ZFLOP)	74LS74	14	7
P53 (ZPORT)	74LS02	14	7
P44 (ZFAST)	74LS367	20	10
PC VERSION ONLY:			
Z25	74LS04	14	7

PART	VALUE	NOTES
C1	330 pf	
C2	33 nf (.033 mf)	
R1	10k (10,000 ohms)	
VCR1	100k (100,000 ohms)	Variable

Miscellaneous:  
One 16-pin wire-wrap integrated circuit socket for piggybacking the PC board version onto Z45.  
No socket is needed for the hard-wired version.

PARTS LIST  
HIGH-RESOLUTION GRAPHICS BOARD  
POWER SUPPLY

PART	VALUE	NOTES
C1	4700 mf, 35 volts	
C2	100 mf, 25 volts	
C3	100 nf (0.1 mf)	
C4	10,000 mf, 35 volts	
C5	100 mf, 16 volts	
C6	100 nf (0.1 mf)	
C7	470 mf, 16 volts	
C8	100 mf, 16 volts	
C9	100 nf (0.1 mf)	
R1	220 ohms, 1/2 watt	

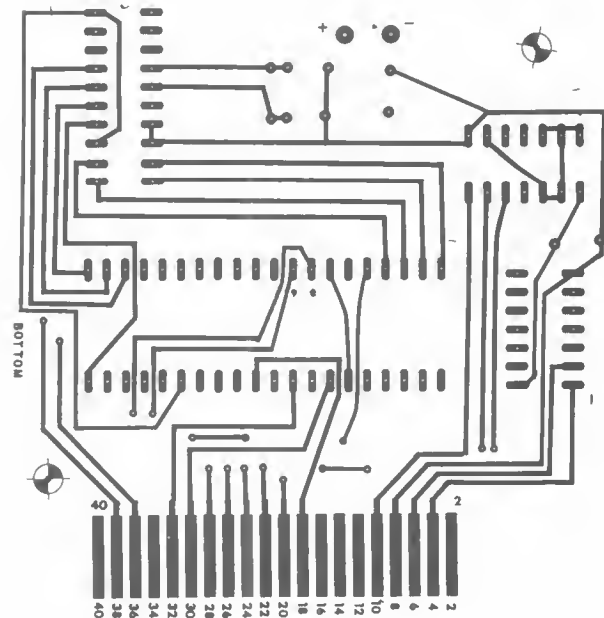
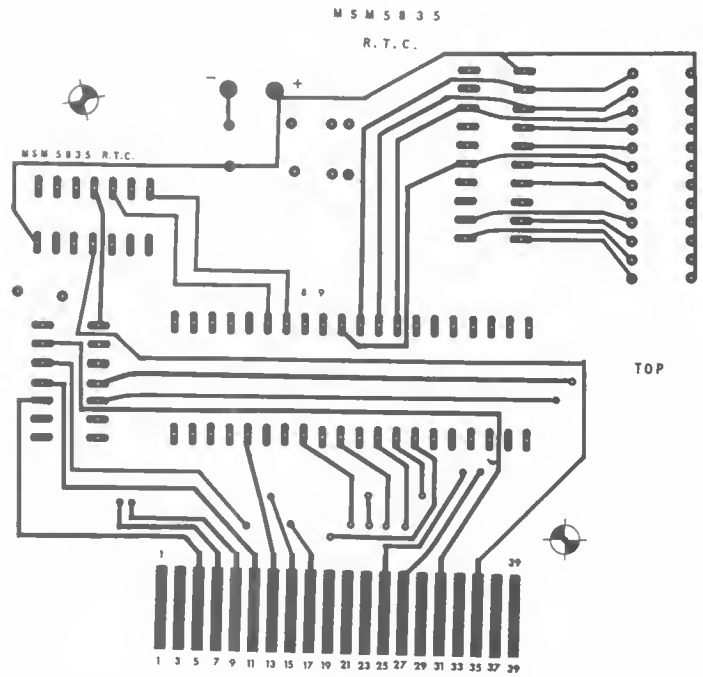
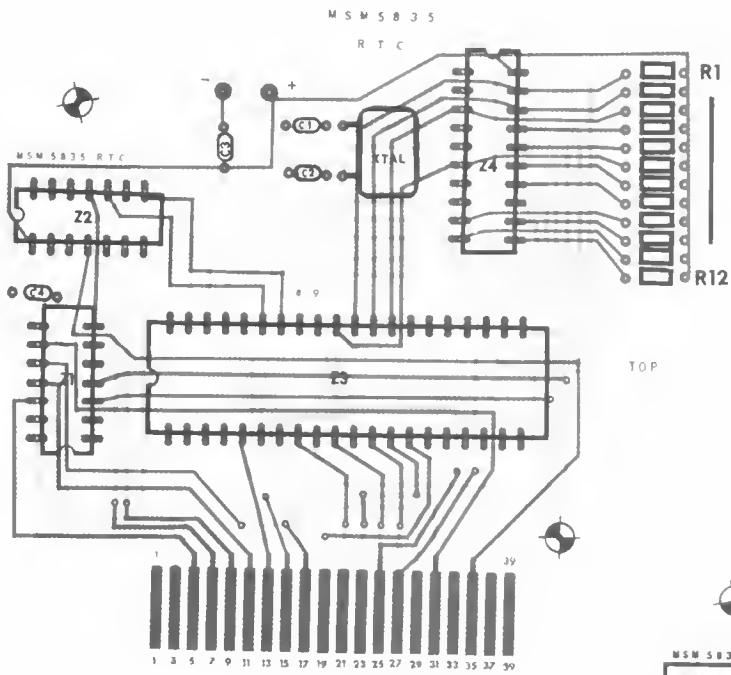
PARTS LIST  
HIGH-RESOLUTION GRAPHICS BOARD

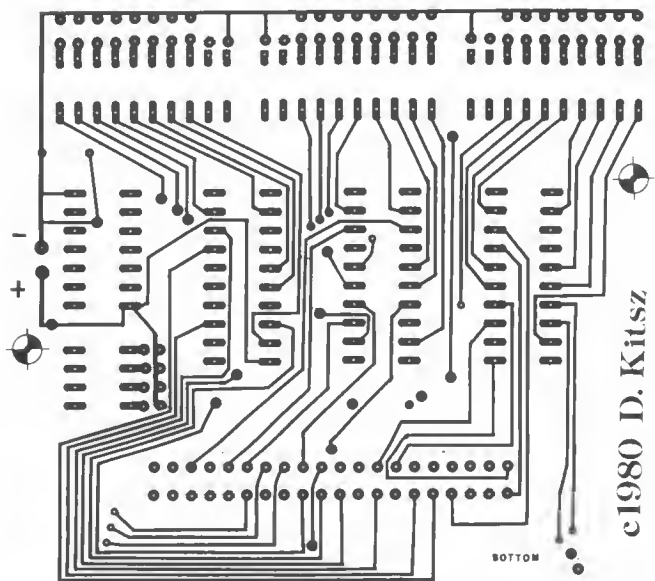
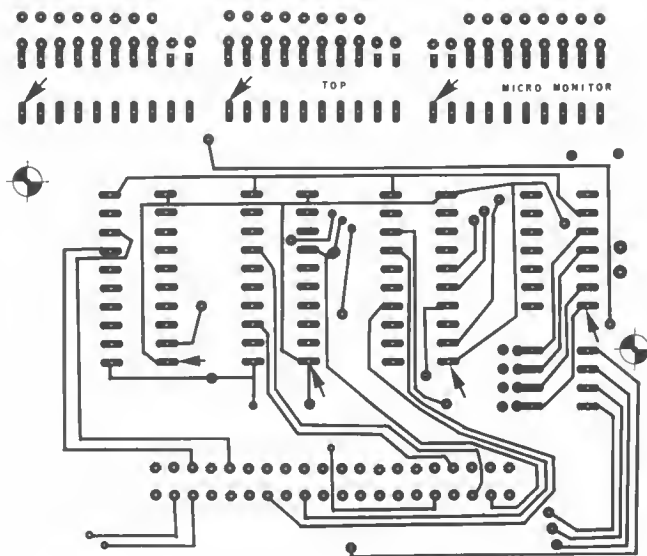
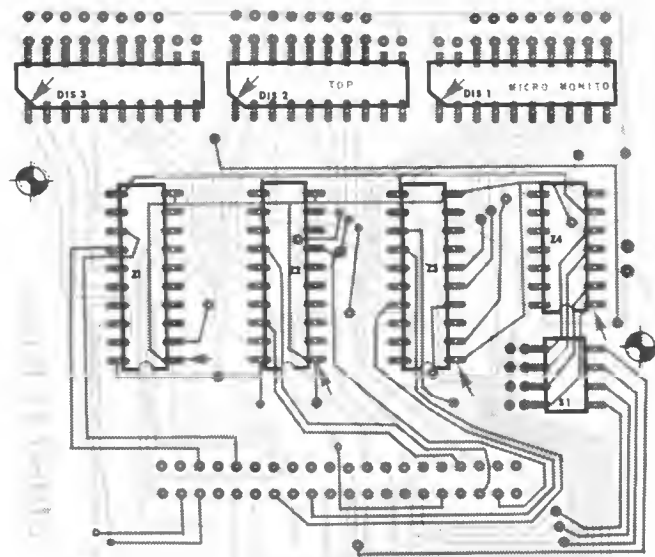
IC	TYPE	+5 VOLTS	GROUND
Z1	74LS04	14	7
Z2	74LS92	5	10
Z3	74LS74	14	7
Z4	74LS93	5	10
Z5	74LS93	5	10
Z6	74LS93	5	10
Z7	74LS93	5	10
Z8	74LS11	14	7
Z9	74LS02	14	7
Z10	74LS157	16	8
Z11	74LS157	16	8
Z12	74LS157	16	8
Z13	74LS157	16	8
Z14	74LS74	14	7
Z15	74LS00	14	7
Z16	74LS174	16	8
Z17	74LS166	16	8
Z18	74LS157	16	8
Z19	74LS157	16	8
Z20	74LS157	16	8
Z21	MK4116	9	16
Z22	MK4116	9	16
Z23	MK4116	9	16
Z24	MK4116	9	16
Z25	MK4116	9	16
Z26	MK4116	9	16
Z27	74C04	14	7
Z28	74C04	14	7
Z29	74C00	14	7
Z30	75452	8	4
Z31	74LS00	14	7

Note that Z21 through Z26 also need -5 volts on pin number 1, and +12 volts on pin number 8.

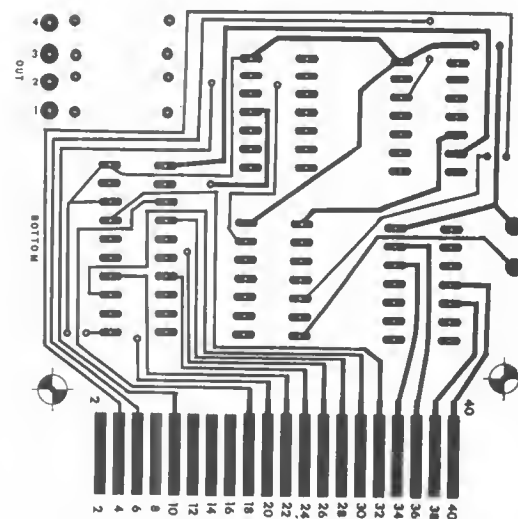
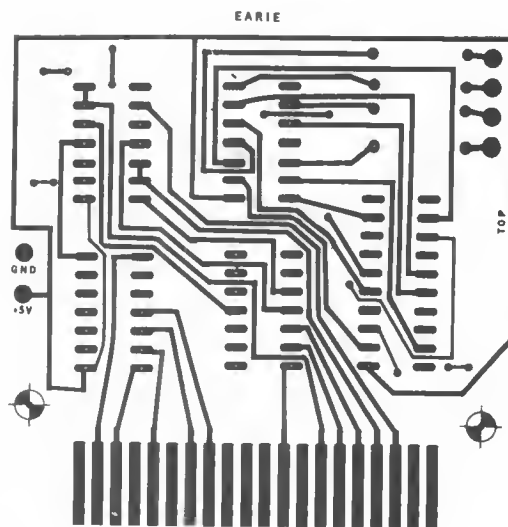
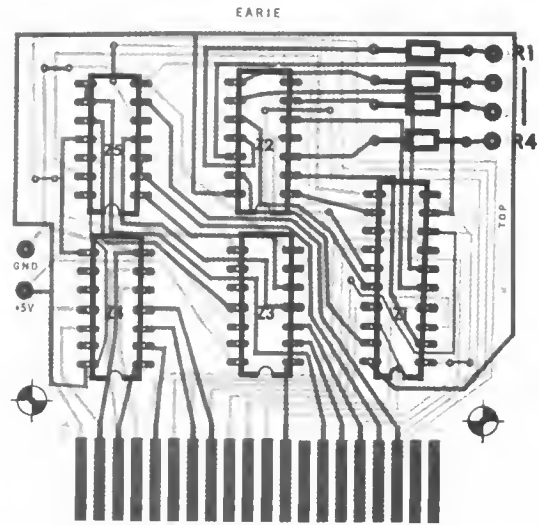
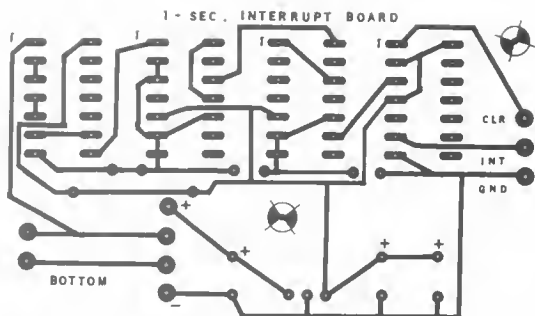
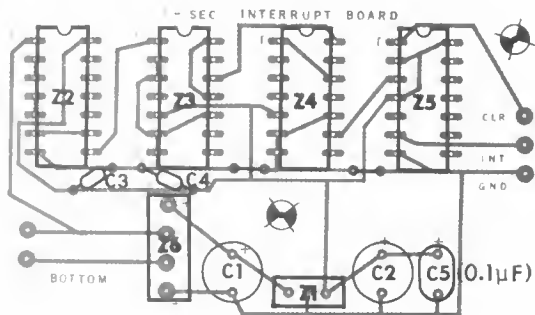
PART	VALUE	NOTES
C1*	47 pf	Miniature variable
C2	10 nf (.01 mf)	
C3	10 mf, 16v	
C4	330 pf	10% or better
C5	47 nf (.047 mf)	10% or better
C6	750 pf	10% or better
C7	22 nf (.022 mf)	10% or better
Q1	2N3904	NPN switching
Q2	2N3904	NPN switching
R1	1k0 (1000 ohms)	
R2	1k0 (1000 ohms)	
R3	1k0 (1000 ohms)	
R4*	910R (910 ohms)	
R5*	910R (910 ohms)	
R6	1k8 (1800 ohms)	
R7	47R (47 ohms)	
R8	270R (270 ohms)	
R9	120R (120 ohms)	
R10	330R (330 ohms)	
R11	75R (75 ohms)	
R12	10k (10,000 ohms)	
R13	10k (10,000 ohms)	
VCR1	100k (100,000 ohms)	Variable
VCR2	100k (100,000 ohms)	Variable
X1*	10.6445 MHz	

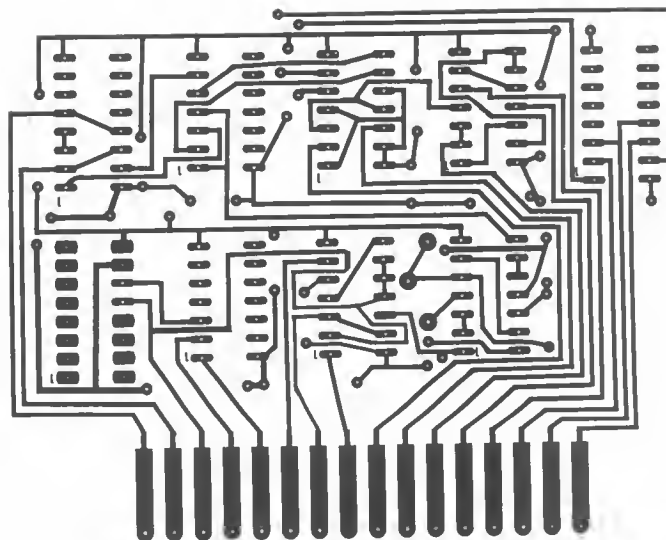
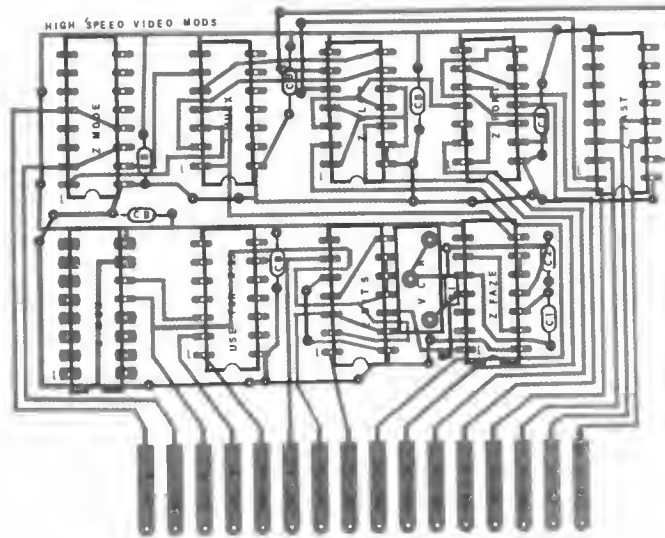
\*Not needed if computer's internal clock is used.





c1980 D. Kitz

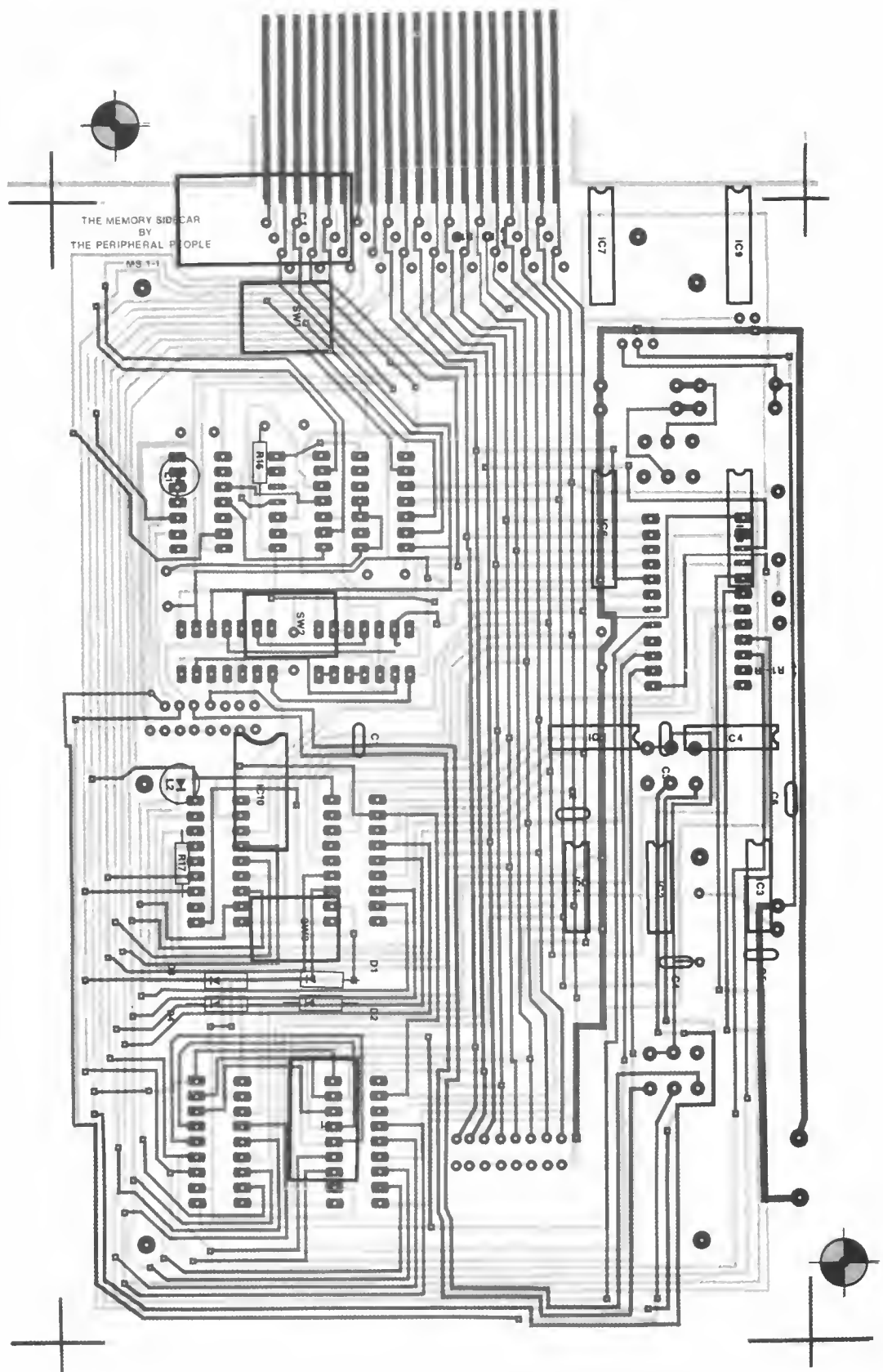


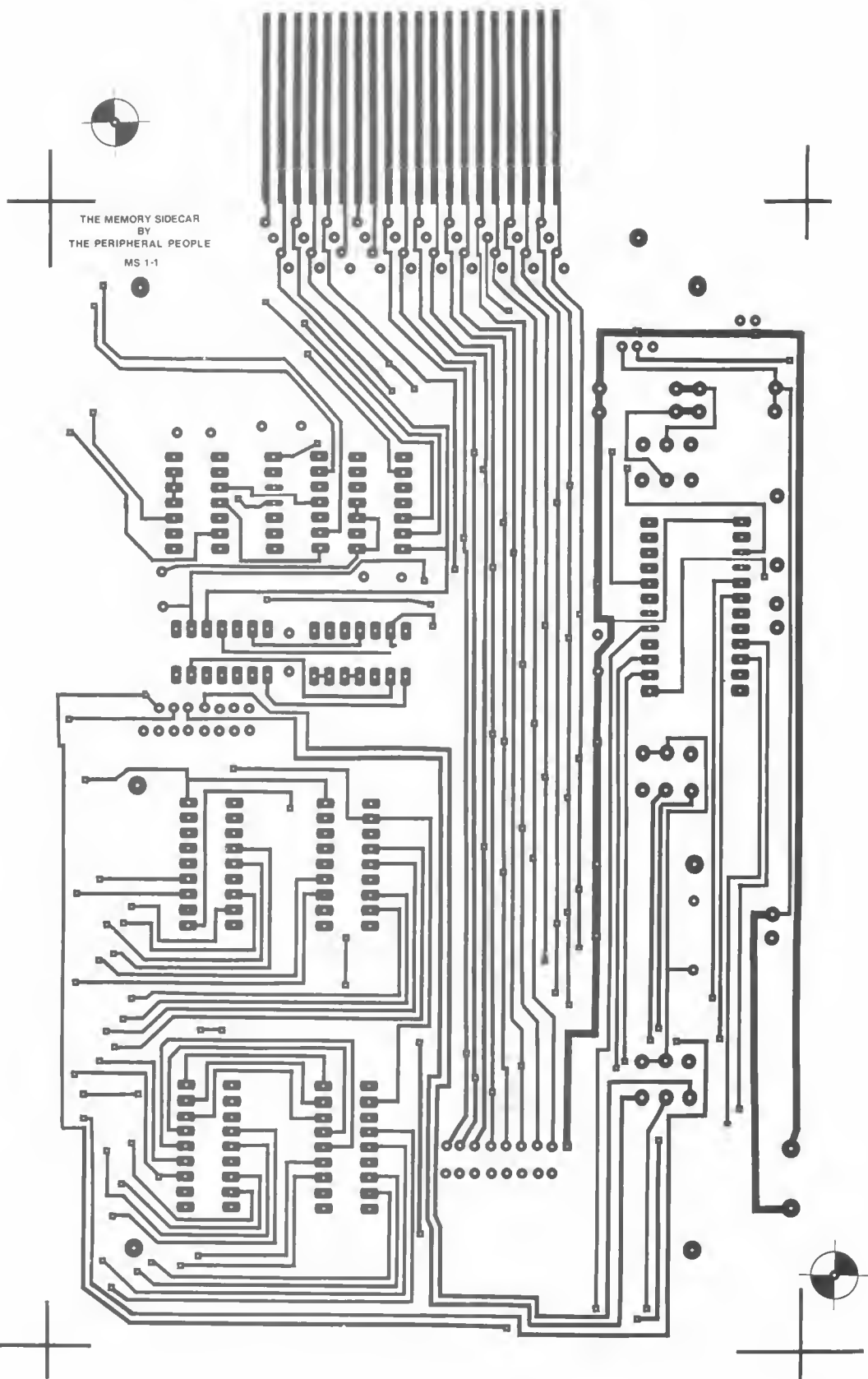


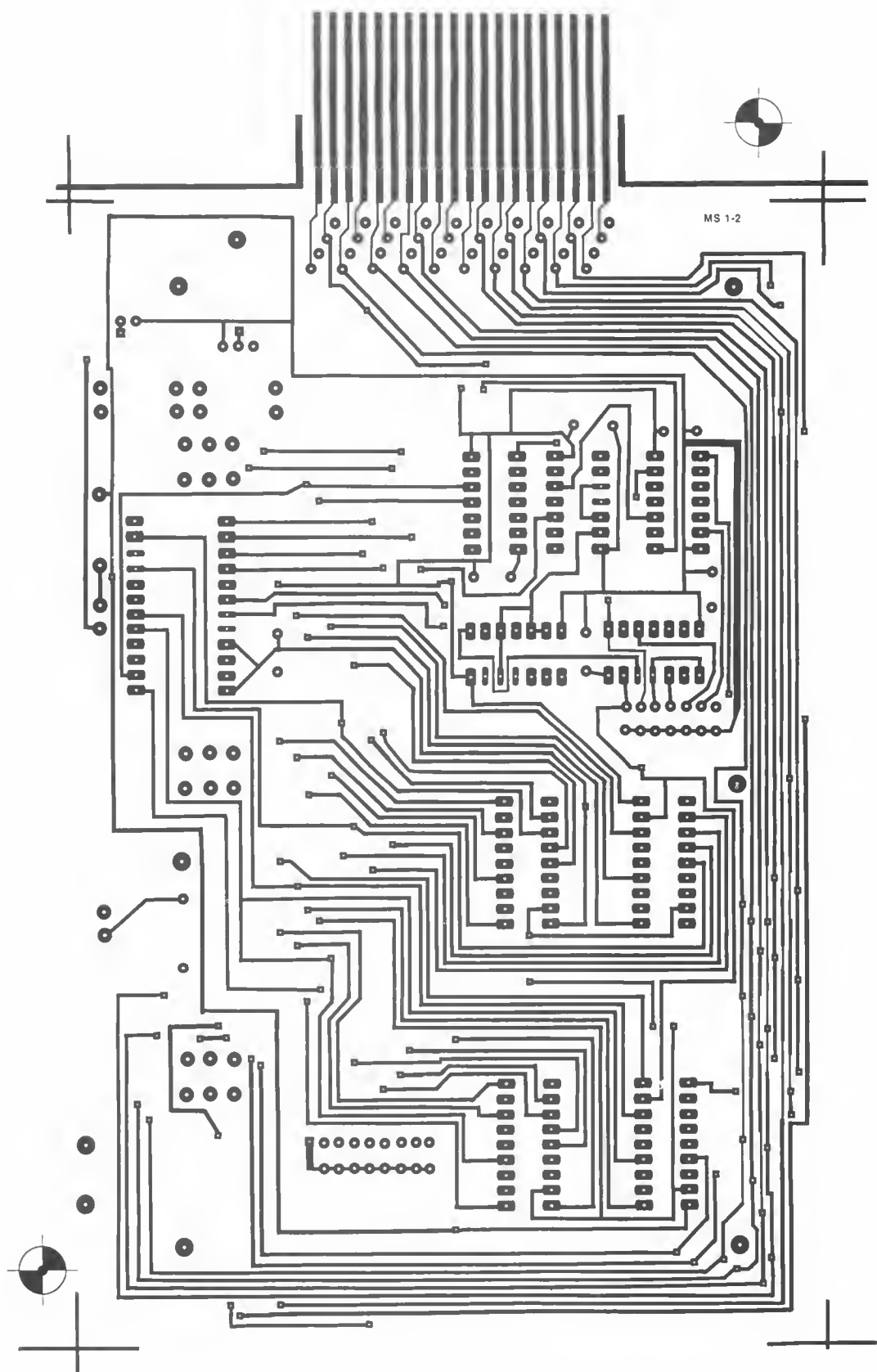


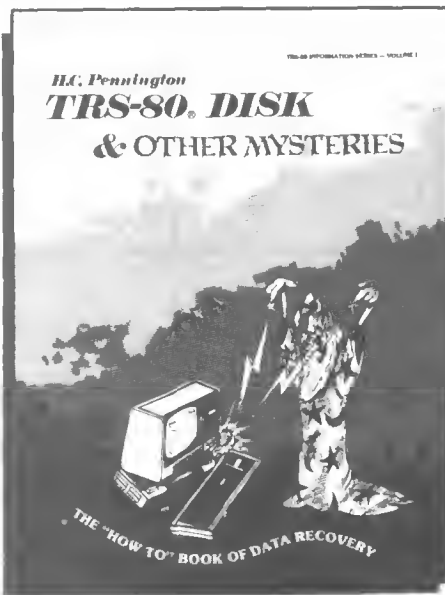












## TRS-80. DISK AND OTHER MYSTERIES

by H. C. Pennington

*TRS-80 Disk and Other Mysteries* is the definitive fix-it book for disk users. More than 130 pages of easy to read entertaining and immensely useful information. Find out how to recover disk files, the layout of information on disks, memory maps, problem solutions . . . the list goes on!

Many readers have saved days of work by recreating disk files that were unreadable. *TRS-80 Disk and Other Mysteries*, which has received favorable reviews in several magazines, is yours for only \$22.50 (plus \$3.00 shipping. CA residents add \$1.35 sales tax. Canada add 20% for exchange rate.) TRS-80 is a trademark of Tandy



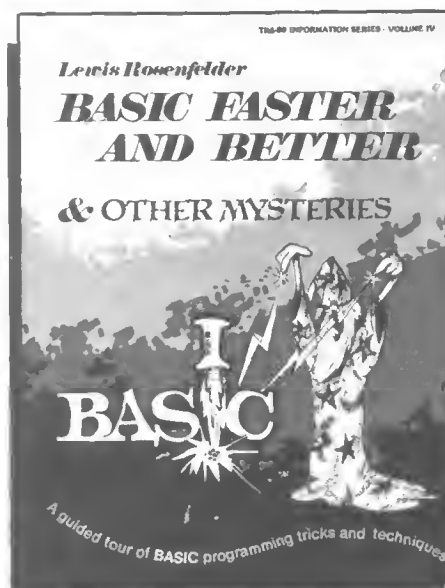
## MICROSOFT. BASIC DECODED AND OTHER MYSTERIES

by James Farvour

*Microsoft Basic and Other Mysteries* is the definitive guide to your Level II ROMs. With more than 7,000 lines of detailed comments and 6 additional chapters packed with information, it is easily the biggest and best book about the Level II ROMs available.

Exploit the full power of Microsoft Basic with the aid of more than 300 pages of tested examples, explanations, and detailed comments. *Microsoft Basic and Other Mysteries* is yours for only \$29.95 (plus \$3.00 shipping. CA residents add \$1.80 sales tax. Canada add 20% for exchange rate.)

\* T.M. Microsoft



## BASIC FASTER AND BETTER AND OTHER MYSTERIES

by Lewis Rosenfelder

Basic is not nearly as slow as most programmers think. *Basic Faster and Better* shows you how to supercharge your BASIC with almost 300 pages of fast, functions and subroutines.

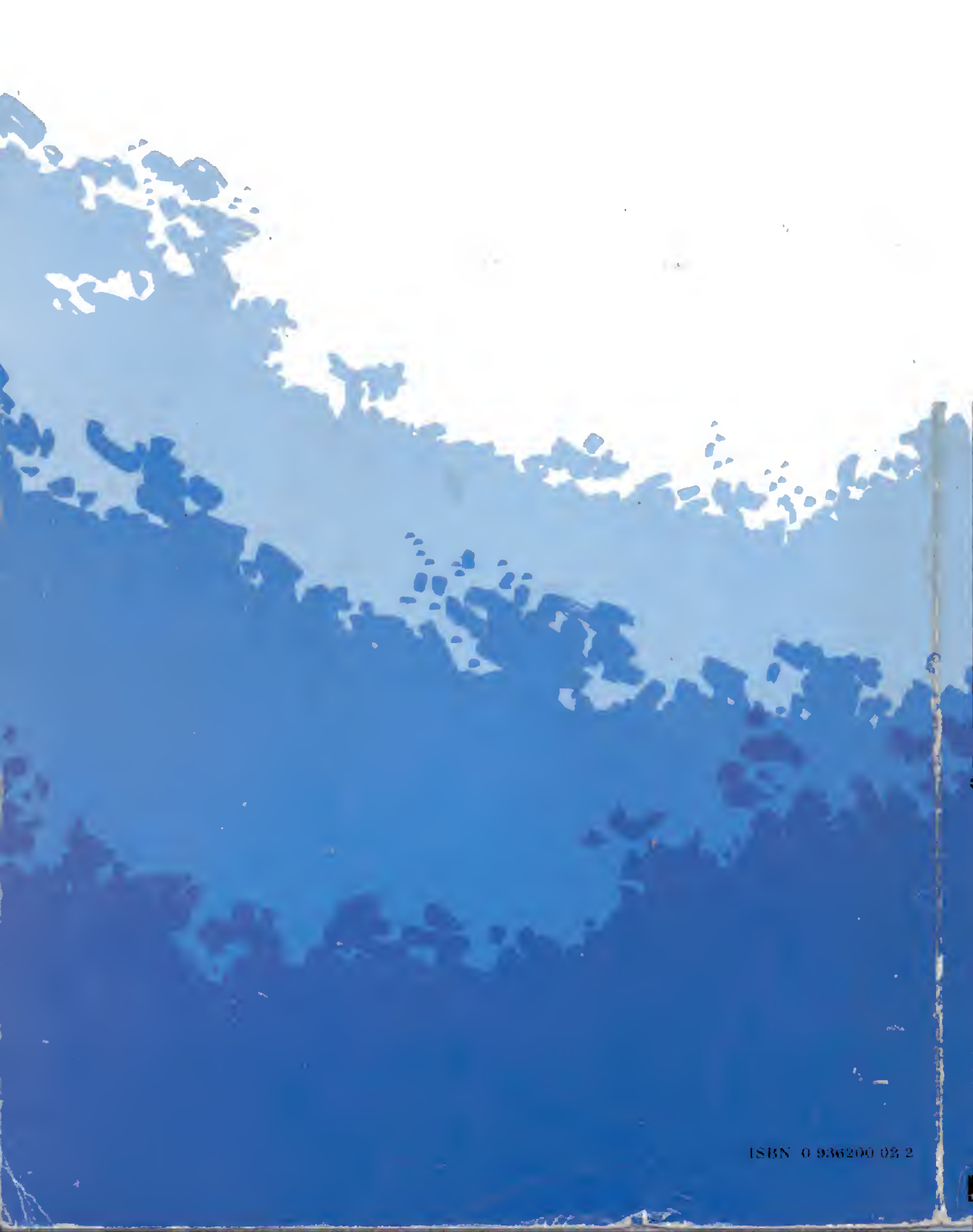
You won't find any trivial poorly designed "check-book balancing" programs in this book — it's packed with *useful* programs.

Tutorial for the beginner, instructive for the advanced, and invaluable for the professional, this book doesn't just talk . . . it shows how! All routines are also available on disk, so that you can save hours of keyboarding and debugging.

The #1 disk *BFBDEM* contains all the demonstration programs, and #2 disk *BFBLIB* has all the library functions.

*Basic Faster and Better* is \$29.95, and the two disks are \$19.95 each (plus \$3.00 shipping for book and disks together or separate. CA residents add \$1.80 sales tax for the book, disks add \$1.20 each. Canada add 20% for exchange rate.)

OTHER  
MYSTERIES BOOKS  
FROM I.J.G.



ISBN 0 936200 02 2